UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA ELÉTRICA

Emanueli Santos da Silva

DESENVOLVIMENTO DE UM SISTEMA EMBARCADO COM O DISPOSITIVO ODROID-XU4 PARA INTEGRAÇÃO SENSORIAL DE UM VEÍCULO TERRESTRE NÃO TRIPULADO

Santa Maria, RS
2018

Emanueli Santos da Silva

DESENVOLVIMENTO DE UM SISTEMA EMBARCADO COM O DISPOSITIVO ODROID-XU4 PARA INTEGRAÇÃO SENSORIAL DE UM VEÍCULO TERRESTRE NÃO TRIPULADO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica da Universidade Federal de Santa Maria (UFSM), como requisito parcial para a obtenção do grau de **Bacharel em Engenharia Elétrica.**

Orientador: Prof. Dr. Andrei Piccinini Legg

Emanueli Santos da Silva

DESENVOLVIMENTO DE UM SISTEMA EMBARCADO COM O DISPOSITIVO ODROID-XU4 PARA INTEGRAÇÃO SENSORIAL DE UM VEÍCULO TERRESTRE NÃO TRIPULADO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica da Universidade Federal de Santa Maria (UFSM), como requisito parcial para a obtenção do grau de **Bacharel em Engenharia Elétrica.**

Aprovado em 18 de dezembro de 2018:

_____
Andrei Piccinini Legg, Dr. (UFSM)
(Presidente/Orientador)

_____
Daniel Fernando Tello Gamarra, Dr. (UFSM)

_____
Natanael Rodrigues Gomes, Dr. (UFSM)

Santa Maria, RS
2018

DEDICATÓRIA

Aos meu queridos avós Firmina (in memorian), Ordalina, Liberal e João, pelos quais sou grata pelo exemplo de vida, ensinamentos e dedicação a mim.

Ao Glenio e à Maria, meus amados pais que sempre apoiaram e incentivaram minhas escolhas. Serei eternamente grata aos seus esforços para a concretização de minha formação profissional.

AGRADECIMENTOS

Ao professor Dr. Andrei, pelo tempo, dedicação e paciência destinados à minha orientação.

Ao professor Dr. Gamarra, por possibilitar a realização deste trabalho.

A todos que de alguma forma contribuíram para a concretização deste projeto, em especial, aos integrantes do GARRA e NUPEDEE.

Aos amigos Eduardo, Marina, Juliana, Gustavo, Élen e Larissa, agradeço pelo companheirismo.

RESUMO

DESENVOLVIMENTO DE UM SISTEMA EMBARCADO COM O DISPOSITIVO ODROID-XU4 PARA INTEGRAÇÃO SENSORIAL DE UM VEÍCULO TERRESTRE NÃO TRIPULADO

AUTOR: Emanueli Santos da Silva
ORIENTADOR: Andrei Piccinini Legg

O presente trabalho tem como objeto de estudo um veículo robótico desenvolvido no grupo GARRA, da Universidade Federal de Santa Maria. Como objetivo principal, tem-se a tarefa de adaptar o robô pré-existente, a fim de torná-lo uma plataforma móvel independente, capaz de processar os dados de interação sensorial com o ambiente, simplificando o sistema de comunicação da plataforma como um todo através do uso de um dispositivo embarcado. Espera-se, a partir deste projeto, implementar o dispositivo Odroid-XU4 para desenvolver um sistema de comunicação remota, além de realizar a integração entre *hardware* e *software do robô.* A plataforma robótica Poter disponibilizada pelo GARRA motivou o desenvolvimento deste trabalho, visto que essa apresentava falhas, principalmente, com relação à integração sensorial e processamento de informações. O aprimoramento deste robô consiste no agrupamento de componentes eletrônicos, de maneira que esses trabalhem em conjunto, possibilitando o desenvolvimento de trabalhos futuros de forma simples e acessível. Dada a gama de conceitos a serem aplicados para melhoria do robô móvel, justifica-se a contribuição deste trabalho na formação de Engenheiro Eletricista.

Palavras-chave: Veículo robótico. Hardware. Embarcado.

ABSTRACT

DEVELOPMENT OF NA EMBEDDED SYSTEM WITH THE ODROID-XU4 DEVICE FOR SENSORIAL INTEGRATION OF AN UNCONSCIOUS LAND VEHICLE

AUTHOR: Emanueli Santos da Silva
ADVISOR: Andrei Piccinini Legg

The present work has as object of study a robotic vehicle developed in the group GARRA, of the Federal University of Santa Maria. The main objective is to adapt the pre-existing robot in order to make it an independent mobile platform capable of processing the data of sensory interaction with the environment, simplifying the communication system of the platform as a whole through the use of an embedded device. From this project, it is hoped to implement the Odroid-XU4 device to develop a remote communication system, as well as to integrate the hardware and software of the robot. The Poter robotic platform provided by GARRA motivated the development of this work, since it presented flaws, mainly in relation to sensory integration and information processing. The improvement of this robot consists of the grouping of electronic components, so that they work together, allowing the development of future jobs in a simple and accessible way. Given the range of concepts to be applied to improve the mobile robot, the contribution of this work in the formation of an Eletrical Engineer is justified.

Keywords: Robotic vehicle. Hardware. Embedded.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GARRA	Grupo de Automação e Robótica Aplicada
ROS	Robot Operating System
WEB	World Wide Web
LIDAR	Light Detection and Ranging
CI	Circuito Integrado
TX	Transmissão
RX	Recepção
PWM	Pulse Width Modulation
ARM	Advanced RISC Machine
I/O	Input/Output
eMMC	Embedded Multimedia Card
SONAR	Sound Navigation and Ranging
TCP/IP	Transmission Control Protocol/Internet Protocol
AC	Corrente alternada
DC	Corrente contínua
rpm	Rotações por minuto

SUMÁRIO

1 INTRODUÇÃO

No mundo contemporâneo destacam-se grandes avanços na área da robótica, dada a crescente demanda por equipamentos seguros e eficientes, bem como as novas necessidades do consumidor. Sistemas robóticos possibilitam um aumento na produtividade, flexibilidade, qualidade e segurança na execução de tarefas em diferentes áreas do conhecimento, justificando o desenvolvimento de trabalhos nesta área.

A robótica se apresenta como uma ciência, dedicada ao desenvolvimento de dispositivos capazes de executar tarefas de forma automatizada, englobando conceitos de mecânica, elétrica e computação (PIO, CASTRO e C., 2006). Seu avanço implica em uma revolução em atividades de risco à vida humana, como atividades de resgate, incêndios e desastres ambientais. Ainda, tem-se as pesquisas com foco em veículos de transporte terrestre autônomos, as quais estão em avanço contínuo, com testes reais sendo realizados por diferentes empresas.

Um dos desafios da robótica é integrar as informações enviadas pelos sensores que interagem com o ambiente, bem como processá-las, de modo a gerar comandos e controlar dispositivos atuadores. É importante ressaltar que a resposta deve ocorrer de forma precisa, a fim de evitar riscos ao robô e àqueles que o cercam (DENIS FERNANDO WOLF, 2009). De forma a minimizar tal problema é possível implementar o controle do robô a partir de um dispositivo embarcado, o qual possibilita a comunicação com diversos sistemas, facilitando o envio de comandos, tomada de decisões, bem como o cruzamento de informações. Assim, se aumenta a flexibilidade da plataforma, visto que o processamento de dados sensoriais pode ocorrer de maneira mais simples e eficiente.

Neste contexto, este trabalho tem como objeto de estudo um veículo robótico desenvolvido no grupo GARRA, da Universidade Federal de Santa Maria.

1.1 OBJETIVO GERAL

Como objetivo geral, tem-se a tarefa de adaptar o robô pré-existente, a fim de torná-lo uma plataforma móvel independente, capaz de processar os dados de interação sensorial com o ambiente, simplificando o sistema de comunicação da plataforma como um todo através do uso de um dispositivo embarcado.

1.1.1 OBJETIVOS ESPECÍFICOS

- Implementar um dispositivo embarcado a um robô pré-existente;
- Realizar a integração de componentes eletrônicos, *hardware* e *software;*
- Projeto de uma placa de circuito impresso para implementação de sensores ultrassônicos;
- Desenvolver um sistema de comunicação entre sensores ultrassônicos e microcontrolador;
- Desenvolver um sistema de comunicação entre microcontrolador e dispositivo embarcado;
- Desenvolver um sistema de comunicação remota;
- Implementar um sensor LIDAR na plataforma robótica;
- Desenvolver um algoritmo para desvio de obstáculos;
- Desenvolver uma interface para envio de comandos;

1.2 JUSTIFICATIVA

A plataforma robótica Poter disponibilizada pelo GARRA motivou o desenvolvimento deste trabalho, visto que essa apresentava falhas, principalmente, com relação à integração sensorial e processamento de informações, ou seja, comunicação *hardware-software*. O aprimoramento deste robô consiste no agrupamento de componentes eletrônicos, de maneira que esses trabalhem em conjunto, possibilitando o desenvolvimento de trabalhos futuros de forma simples e acessível. Dada a gama de conceitos a serem aplicados para melhoria do robô móvel, justifica-se a contribuição deste trabalho na formação de Engenheiro Eletricista.

1.3 ORGANIZAÇÃO DO TRABALHO

No capítulo 2 é apresentada uma revisão bibliográfica, a fim de situar o leitor, introduzindo conceitos. Já no capítulo 3 são descritas as principais ferramentas utilizadas no desenvolvimento deste trabalho. Após, no capítulo 4 apresenta-se uma visão geral do projeto, na qual o leitor será capaz de compreender o sistema que será discutido e detalhado nos capítulos seguintes. No capítulo 5 apresenta-se a metodologia do projeto, na qual se discute todo o desenvolvimento do projeto,

especificando os procedimentos realizados. Já no capítulo 6 apresentam-se os resultados obtidos a partir da metodologia utilizada. Por fim, no capítulo 7 o trabalho é concluído, expondo os principais pontos observados durante todo o desenvolvimento do trabalho.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo é apresentada uma breve contextualização da robótica e seus avanços, bem como os principais conceitos envolvidos neste projeto, os quais são necessários para um bom entendimento da estrutura geral do sistema e desenvolvimento do mesmo.

2.1 VISÃO GERAL DA ROBÓTICA

Um grande avanço tecnológico se iniciou no final dos anos 50, com desenvolvimento de circuitos integrados. Presentes na maioria dos equipamentos eletrônicos desenvolvidos, possibilitaram a criação de circuitos mais complexos, como o microprocessador, lançado na década de 70 pela INTEL Corporation (ZAMAIA). Ao longo dos anos os componentes eletrônicos passaram a ter custo e tamanho reduzido, e o consequente avanço das linguagens de programação possibilitaram o desenvolvimento da robótica.

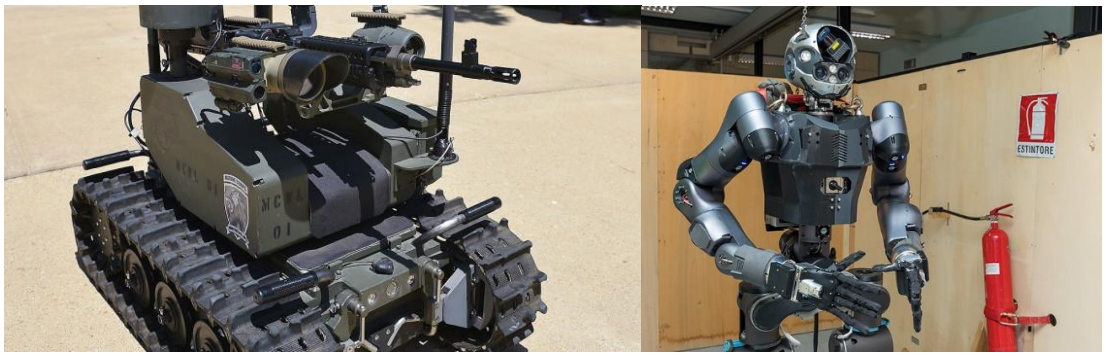
A partir dos anos 80, a robótica se difundiu em diversas aplicações. Na agricultura, agilizando processos de plantio e colheita, nas residências, auxiliando em atividades domésticas, em zonas de risco ao ser humano como limpezas de ambientes tóxicos e trabalhos com explosivos e em diversas outras áreas, como extração de petróleo em alto-mar, onde veículos remotamente operados trabalham em profundidades de até sete mil metros, para evitar e controlar vazamentos (ZAMAIA). O setor industrial é uma área em que o emprego da robótica teve um grande impacto, dada a necessidade de aperfeiçoar o sistema de produção fabril tem por consequência a busca por formas de tornar o processo de fabricação mais eficiente e produtivo. Outra área de destaque da utilização da robótica é a medicina, onde procedimentos cirúrgicos são realizados com assistência de equipamentos, possibilitando uma maior segurança na execução dos processos. Em serviços de segurança, observa-se a crescente evolução de equipamentos que auxiliam, inclusive na defesa nacional: como exemplo tem-se os drones, utilizados para patrulhar as fronteiras da Amazônia. Ainda, pode-se destacar o desenvolvimento de veículos autônomos, os quais visam substituir condutores humanos por um sistema de controle computacional capaz de interagir com ambiente. Nota-se a influência da robótica em

nosso cotidiano, onde a dinâmica do mercado de trabalho é alterada cada vez mais pela automatização de processos em diversas áreas do conhecimento.

2.2 ROBÓTICA MÓVEL

Dispositivos mecânicos dotados de um sistema de locomoção capazes de navegar em um determinado ambiente, robôs móveis permitem a execução de uma tarefa mediante a interpretação de dados adquiridos a partir da leitura do mundo externo, sensores e estado atual do veículo (SECCHI, 2008). Com aplicações em diversas áreas de conhecimento, destacam-se a realização de atividades nocivas à saúde humana ou, de difícil execução, como transporte de cargas perigosas, aplicações agrícolas, militares, bem como missões de busca e resgate. A Figura 1 exemplifica alguns robôs móveis:

Figura 1 - Robô móvel aplicação militar (à esquerda); atividade de resgate (à direita)



Fonte: (GOOGLE IMAGENS, 2018).

A seguir citam-se os principais componentes de um robô, essenciais para seu funcionamento:

- Manipulador: parte mecânica móvel do robô; geralmente garras ou rodas;
- Atuador: dispositivo responsável pela movimentação do manipulador, correspondem à interface de atuação no ambiente externo; podem ser, por exemplo, motores elétricos;
- Sensor: componente sensorial, possibilita a percepção do mundo físico;

- Controlador: unidade central, responsável pelo controle de todos os dispositivos e planejamento das ações da máquina; geralmente um microcontrolador ou computador;
- Fonte de energia: essencial para o funcionamento do controlador, tipicamente uma bateria.

Robôs móveis autônomos, são agentes programáveis capacitados a tomarem decisões com ou sem auxílio externo. Para tanto, além dos sensores e atuadores se faz necessária uma unidade de processamento embarcada no robô (SECCHI, 2008). Visto que permitem uma conexão inteligente entre percepção do ambiente e realização de uma ação resposta, o grau de autonomia está relacionado com a capacidade de abstração de estímulos externos e conversão em ordens, as quais aplicadas aos atuadores resultam na eficácia de uma determinada tarefa.

3 COMPONENTES UTILIZADOS

A seguir são descritos os principais elementos de *hardware* e *software* utilizados no trabalho, de forma a possibilitar ao leitor uma visão detalhada das especificações de cada componente.

3.1 MICROCONTROLADOR

Sistema que consiste em um circuito integrado constituído de processador e circuitos periféricos, como memória de dados, memória de programa, pinos de *Input/Output*, *timers* e conversores A/D (analógico/digitais) (BORBA, 2017). Visto que apresentam apenas um encapsulamento, microcontroladores são aplicados no controle de processos reduzindo sua complexidade.

3.1.1 Arduino MEGA 2560

Criado em 2005, o Arduino é uma plataforma eletrônica *open source* baseada em *hardware* e *software* flexíveis e de fácil utilização (ARDUINO, 2018). Pode facilmente ser conectado à um computador e programada via IDE (*Integrated Development Environment*) por conexão USB, a partir de linguagem baseada em C.

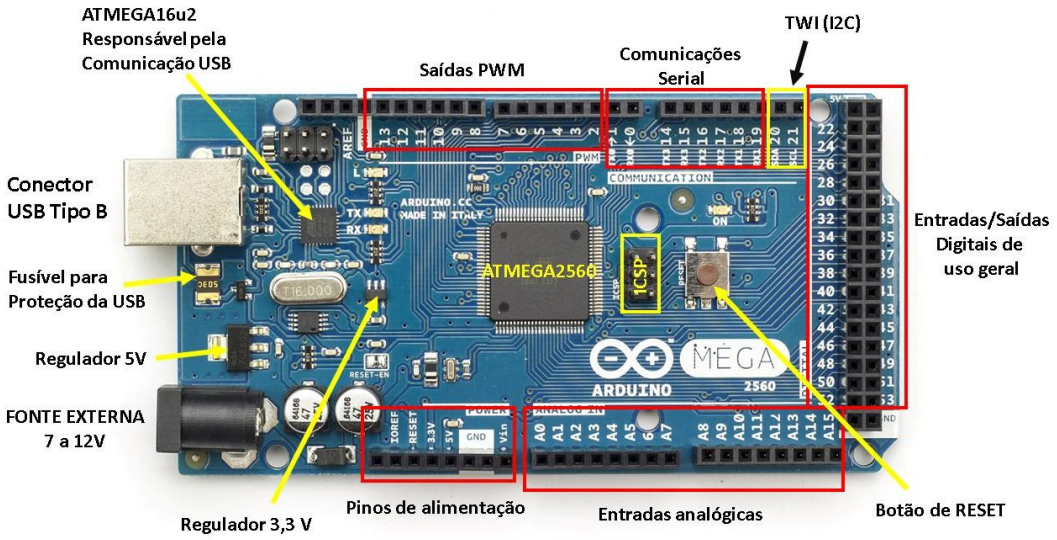
Arduino MEGA 2560 é um modelo da Arduino baseado no microcontrolador ATmega2560, o qual possui 256KB de memória *flash* para armazenamento de código (dos quais 8KB é usado para o *bootloader*), 8KB de SRAM e 4KB de EEPROM (o qual pode ser lido e escrito com a EEPROM). A alimentação pode ser realizada via USB ou fonte externa, sendo àquela processada diretamente pela USB (sem ser estabilizada por regulador de tensão), e essa efetivada a partir de conector *jack* com valor de tensão entre 6V a 20V. Entretanto, ao se utilizar uma fonte externa com tensão abaixo de 7V, a tensão de funcionamento, de 5V, pode ficar instável e, quando alimentada com tensão acima de 12V, o regulador de tensão da placa pode sobreaquecer e danificar o dispositivo. Sendo assim, recomendam-se valores de tensão entre 7V e 12V para fontes externas, os quais são regulados pelo CI OnSemi NCP1117.

O circuito de conexão à USB apresenta alguns componentes que protegem a porta USB do computador em caso de alguma anormalidade: um fusível de 500mA

impede que a porta USB queime, caso ocorra algum problema de projeto ou uma falha no circuito quando a placa estiver conectada ao PC; um ferrite impede que ruídos externos entrem no circuito da placa Arduino, através do GND (SOUZA, 2014).

Ademais, o dispositivo conta com um circuito para comutar a alimentação automaticamente entre a tensão da USB e a tensão da fonte externa. Assim, caso haja uma tensão no conector DC e a placa seja conectada à USB, a tensão de 5V será proveniente da fonte externa e a USB servirá apenas para comunicação. A Figura 2 mostra a placa Arduino MEGA, destacando as principais conexões dessa, bem como a localização do microcontrolador ATmega2560:

Figura 2 - Placa Arduino MEGA 2560



Fonte: (SOUZA, 2014).

Conforme Figura 2, nota-se que o dispositivo possui sete pinos de alimentação, os quais serão listados a seguir com suas respectivas funções:

IOREF – permite que *shields* selecionem uma interface apropriada a partir de uma referência de tensão, assim, possibilita a adaptação entre 3,3V ou 5V;

RESET – utilizado para um *reset* externo da placa Arduino;

3,3V – fornece tensão de 3,3V para alimentação de *shields* e módulos externos, com corrente máxima de 50mA;

5V – fornece tensão de 5V para alimentação de *shields* e circuitos externos.

GND – pinos de referência, *ground*.

VIN – permite alimentação da placa através de bateria externa (quando se utiliza conector *jack* a tensão da fonte está nesse pino).

O microcontrolador ATMEL ATMEGA16U2 funciona como interface USB para comunicação com o computador, possibilitando o *upload* do código binário gerado após a compilação do programa feito pelo usuário. A comunicação entre placa e computador é indicada por dois *leds* (TX, RX), controlados pelo software do microcontrolador, indicando o envio e recepção de dados. Este microcontrolador possui um cristal externo de 16 MHz. A conexão com o ATMEL ATmega2560 é feita pelo canal serial desses microcontroladores.

Além disso, há uma conexão do pino 13 do ATMEGA16U2 ao circuito de RESET do ATMEGA2560, o que implica na entrada automática no modo *bootloader* quando é pressionado o botão *upload* na IDE. Isso não era possível nas primeiras placas Arduino, nas quais era necessário pressionar o botão de RESET antes de fazer o *upload* na IDE (SOUZA, 2014).

O dispositivo Arduino MEGA 2560 possui 16 entradas analógicas, 54 pinos de entradas e saídas digitais, onde 15 destes podem ser utilizados como saídas PWM e 4 portas de comunicação serial, conforme projeto, através das funções pinMode(), digitalWrite(), e digitalRead(), determinadas na IDE. Os pinos operam com tensão de 5V e podem fornecer ou drenar até 40mA. Cada pino possui resistor de *pull-up* interno que pode ser habilitado por *software*. As entradas analógicas (pinos A0 a A15), permitem a conversão AD com uma resolução de 10 bits, ou seja, o valor é convertido entre 0 e 1023. Por padrão a tensão de referência é conectada a 5V, porém, é possível modificá-la através do pino AREF e a função analogReference() (ARDUINO, 2018).

A Tabela 1 apresenta um resumo das principais características da placa discutidas anteriormente.

Tabela 1 - Especificações Arduino MEGA 2560

Arduino MEGA2560	
Microcontrolador	ATmega2560
Tensão de operação	5V
Tensão de entrada (recomendada)	7-12V
Tensão de entrada (limite)	6-20V
Pinos digitais I/O	54 (dos quais 15 são saídas PWM)
Pinos analógicos I/O	16
Corrente CC por pino I/O	20mA
Corrente CC por pino 3,3V	50mA
Memória flash	256KB dos quais 8KB são usados para *bootloader*
SRAM	8KB
EEPROM	4KB
Clock	16MHz
LED_BUILTIN	13
Comprimento	101,52mm
Largura	53,3mm
Peso	37g

Fonte: (ARDUINO, 2018)

3.2 SINGLE BOARD COMPUTER (SBC)

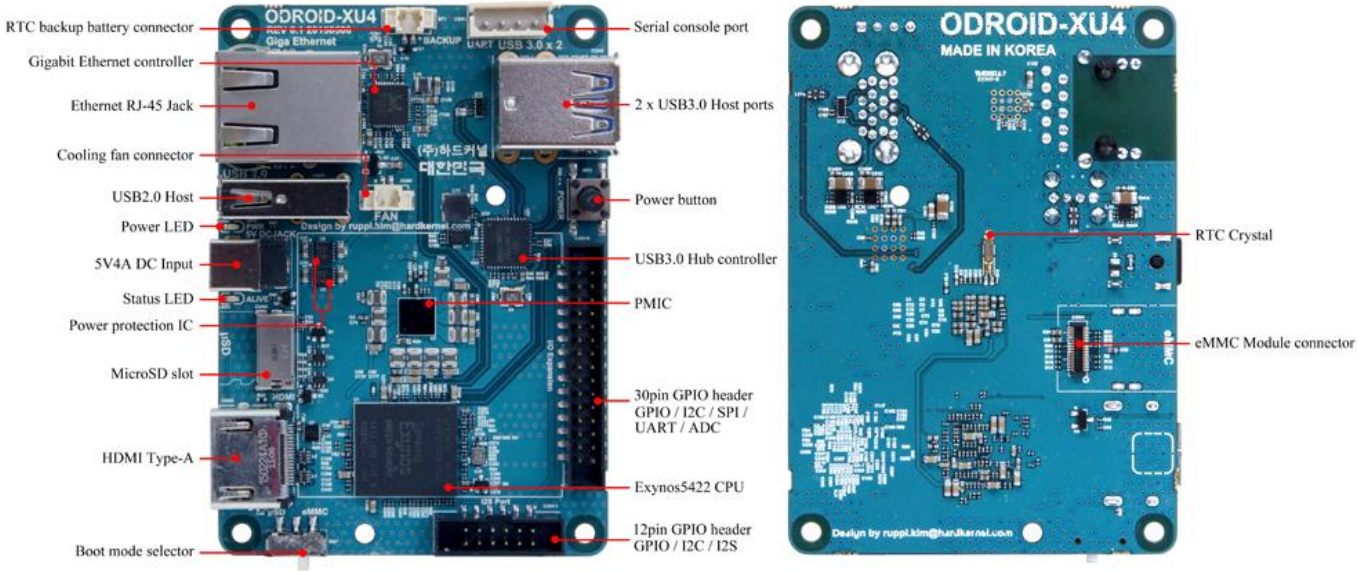
Sistema computacional formado por uma única placa de circuito impresso que contém todos os componentes eletrônicos necessários para seu funcionamento. Apresentam custo e tamanho reduzidos, além de baixo consumo de energia, quando comparados aos *notebooks* e *desktops*.

3.2.1 Odroid-XU4

Lançado em 2015, este embarcado da Hardkernel é um *Single Board Computer* acessível de baixo custo que suporta Linux (Ubuntu, Fedora, ARCHLinux, Debian e outros) ou Android como sistema operacional. Com dimensões de 82x58x20mm e 38 gramas (sem *cooler*), é comumente utilizado em sistemas embarcados.

Utiliza o processador ARM da Samsung, Exynos 5422, Octa core Cortex-A15 2GHz e Cortex-A7. Possui 2 GB de RAM LPDDR3, além de processador gráfico Mali-T628 MP6 (OpenGL ES 3.0/2.0/1.1 e OpenCL 1.1 Full profile) 600MHz, 2 portas USB 3.0 Host, 1 porta USB 2.0 Host, porta Gigabit Ethernet, conexão HDMI 1.4a, alimentação 5V, portas GPIO, conexão para serial e botão de *reset* (HARDKERNEL, 2015). A Figura 3 apresenta as características de hardware do dispositivo:

Figura 3. Odroid-XU4

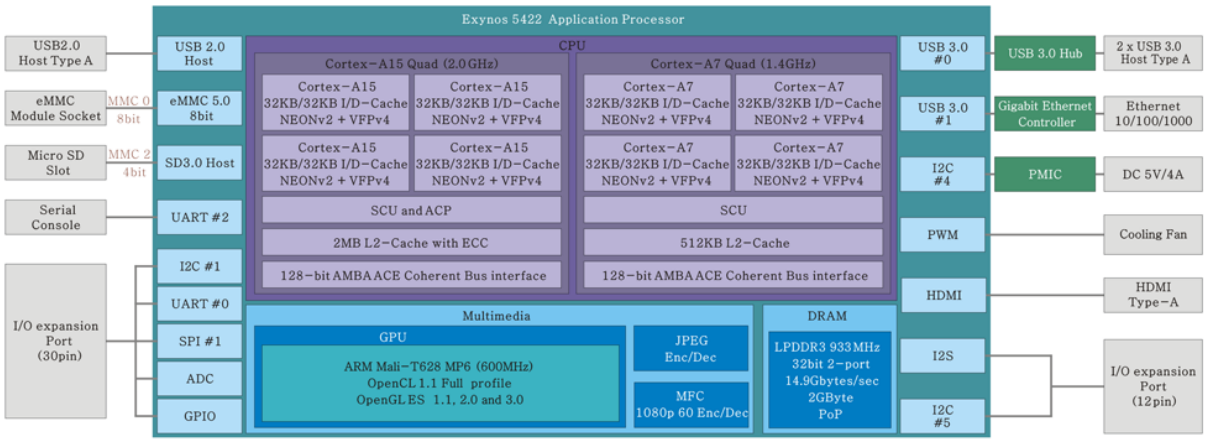


Fonte: (HARDKERNEL, 2015).

O dispositivo apresenta conectores I/O de 30 pinos, que podem ser usados como portas de GPIO/IRQ/SPI/ADC, e 12 pinos, os quais podem ser usados como GPIO/IS2/IC2 para conexão de circuitos eletrônicos. Ambos operam com 1,8V, assim, se o periférico ou sensor a ser utilizado requere uma tensão maior, deve-se utilizar um

shield elevador de tensão. Na Figura 4 pode-se analisar o diagrama de blocos que representa as conexões do embarcado:

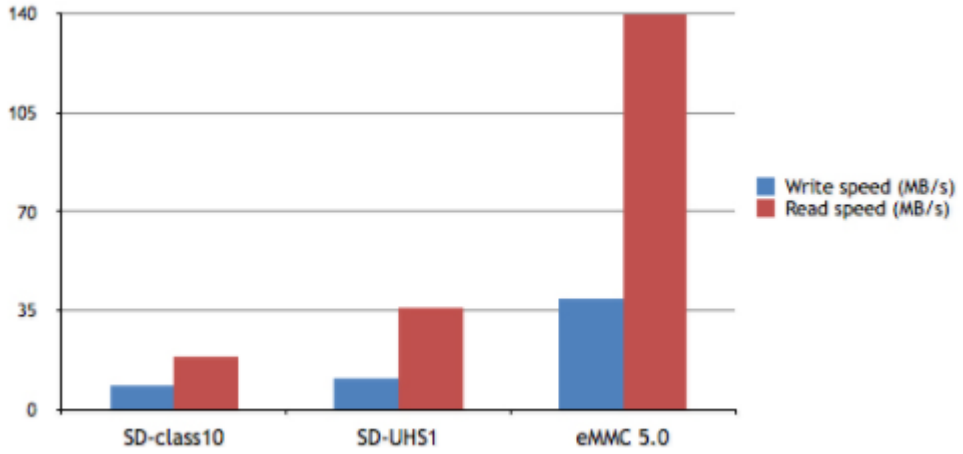
Figura 4. Diagrama de blocos Odroid- XU4



Fonte: (HARDKERNEL, 2015).

Apresenta um consumo médio de 5W, executando o sistema operacional a partir de um cartão microSD ou pelo módulo eMMC. A taxa de transferência de um eMMC é aproximadamente 65MB/s, enquanto que de um cartão microSD UHS-1 é aproximadamente de 30MB/s. O dispositivo é compatível com módulos eMMC de 8GB, 16GB e 64GB. A Figura 5 apresenta um gráfico comparando as velocidades de leitura e escrita dos dispositivos discutidos. Segundo (ROY, 2014), A utilização de um módulo eMMC aumenta a velocidade e capacidade de resposta, semelhante à maneira pela qual a atualização de um SSD em um PC melhora o desempenho em um HDD.

Figura 5. Velocidade de leitura e escrita - SD vs eMMC



Fonte: (HARDKERNEL, 2015).

Um conector *jack*, 5V, com um diâmetro interno de 2,1mm e um diâmetro externo de 5,5mm permite a alimentação do dispositivo. O ODROID-XU4 consome menos de 1A na maioria dos casos, mas pode consumir até 4A quando muitos periféricos USB não energizados forem conectados diretamente à placa principal.

As conexões USB permitem a conexão de periféricos, com mouse, teclado, além de possibilitar o carregamento de dispositivos, como um *smarthphone*, por exemplo (HARDKERNEL, 2015). O conector HDMI Type-A suporta exibição de até 1080p. Para realizar conexões de rede, pode-se utilizar a porta Ethernet, que suporta 10/100/1000Mbps ou um adaptador Wi-Fi via conexão USB. Com intuito de destacar as principais características apresentadas acima, as especificações do dispositivo podem ser visualizadas, resumidamente, na Tabela 2.

Tabela 2 - Especificações Odroid-XU4

ODROID XU4	
Processador	Samsung Exynos544 ARM Cortex™-A15 Quad 2.0GHz/Cortex™-A7 Quad 1.4GHz
Memória	2Gbyte LPDDR3 RAM PoP (750MHz, 12GB/s memory bandwidth, 2x32bit bus)
Hardware Gráfico	Mali™-T628 MP6 OpenGL ES 3.1/3.0/2.0/1.1 e OpenCL 1.2 Full
Áudio	Saída de áudio HDMI Digital
USB3.0 Host	SuperSpeed USB padrão conector tipo A x 2 portas
USB2.0 Host	HighSpeed USB padrão conector tipo A x 1 porta
Display	HDMI 1.4ª com conector Tipo-A
Armazenamento (Facultativo)	Módulo eMMC: armazenamento flash 50. (até 64GByte) Slot para MicroSD (até 182GByte)
Fast Ethernet LAN	10/100/1000Mbps Ethernet com RJ-45 Jack
WiFi (Facultativo)	USB IEEE 802.11 1T1R WLAN com antena (adaptador USB externo)
Interface HDD/SSD SATA (Facultativo)	Adaptador SuperSpeed USB(USB3.0) para Serial ATA3 de 2,5”/3,5” HDD e SSD
Potência de entrada	4,8V~5,2V (recomenda-se 5V/4A)
Sistema Operacional	Ubuntu 16.04 + OpenGLES+OpenCL Linux Kernel 4.14 LTS Android 4.4.2 Kernel LTS 3.10 Android 7.1
Dimensões	83x58x20mm (peso: 38g) sem o cooler

Fonte: (ROY, 2014).

O processador do XU4 pode atingir temperaturas de até 95ºC quando opera em capacidade máxima. A altas temperaturas, o dispositivo reduz sua velocidade de operação e processamento para evitar que a temperatura se eleve. Sendo assim, pode-se utilizar um dissipador de calor, em conjunto com um ventilador, conforme Figura 6, a fim de otimizar o funcionamento do sistema.

Figura 6. Odroid-XU4 com cooler

Fonte: (ROY, 2014).

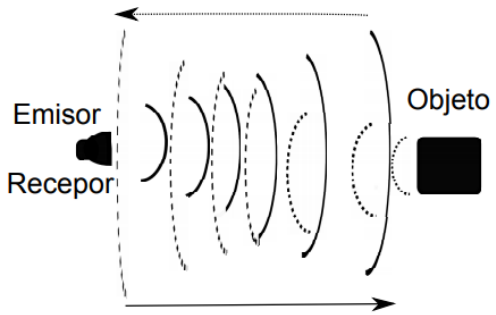
3.3 SENSOR ULTRASSÔNICO

Comumente utilizados na robótica para percepção do ambiente e detecção de obstáculos, sensores ultrassônicos utilizam SONAR como princípio de funcionamento, a qual corresponde à técnica utilizada em submarinos para estimar proximidade à objetos ou presença de outras embarcações. Tal princípio possibilita o cálculo de distância baseando-se no sinal sonoro propagado até o obstáculo.

Consistem basicamente de um transceptor capaz de emitir e detectar som. Este dispositivo emite pulsos sonoros, além da faixa de audição do ouvido humano, de forma cíclica. Quando ocorre um objeto reflete o som emitido um eco é recebido pelo

transceptor e convertido em sinal elétrico. O esquema de funcionamento de um sensor ultrassônico pode ser visualizado Figura 7.

Figura 7 - Funcionamento sensor ultrassônico



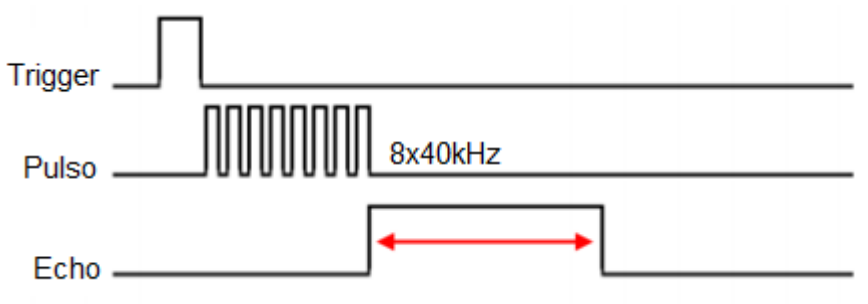
Fonte: (GASPARI, 2014).

A precisão deste sensor é influenciada diretamente por fatores externos. Visto que a velocidade do som pode variar com a temperatura e umidade do ar, condições climáticas diversas podem prejudicar o funcionamento de tal dispositivo. Ademais, os objetos refletem as ondas sonoras de diferentes maneiras, dependendo de sua constituição química: materiais macios, como tecidos por exemplo, absorvem as ondas sonoras e dificultam a reflexão; enquanto metais se mostram bom refletores. Dessa forma, a precisão deste sensor é influenciada pelo tipo de obstáculo a ser detectado (GASPARI, 2014).

3.3.1 HC-SR04

O sensor ultrassônico HC-SR04 é constituído, fisicamente, por 4 pinos: alimentação, *Trigger*, *Echo* e terra. Este dispositivo entra em funcionamento quando o pino *Trigger* é alimentado em nível alto por mais de 10µs. A partir da alimentação deste pino são emitidos 8 pulsos de 40kHz que retornam ao receptor quando encontram um objeto, conforme Figura 8. Durante o tempo de emissão e retorno do sinal o pino *Echo* é mantido em nível alto. Considerando-se a velocidade do som no ar, de 340m/s, a distância do objeto pode ser calculada pela Equação (1):

Figura 8. Diagrama de tempo HC-SR04



Fonte: (WIKI).

$$Distância = {tempo\ do\ pulso\ em\ Echo\ .340m}/{2} \quad (1)$$

Para fins de cálculo, conforme equação acima, o tempo é dividido por dois de forma a considerar efetivamente o caminho percorrido pelo sinal até encontrar o obstáculo (WIKI). A partir da Figura 9 é possível observar o sensor HC-SR04:

Figura 9 - HC-SR04

Fonte: (FLOP).

A tabela abaixo destaca as especificações do dispositivo:

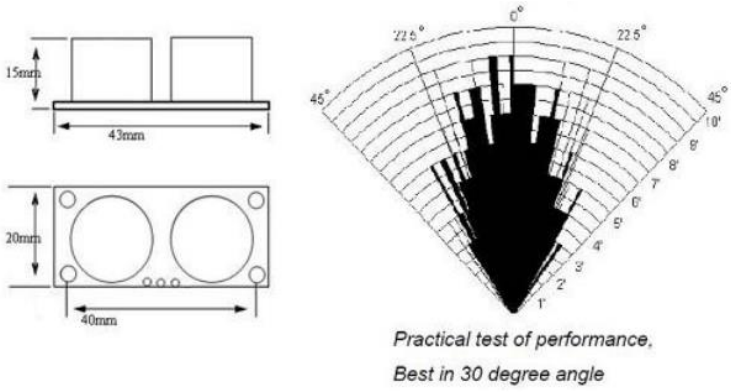
Tabela 3 - Características HC-SR04

Especificações HC-SR04	
Tensão de Alimentação	$5V_{DC}$
Corrente quiescente	< 2mA
Corrente em funcionamento	15mA
Ângulo de medida	< 15°
Distância de detecção	2cm a 400cm
Resolução	3mm
Dimensões	45mm x 20mm x 15mm
Frequência ultrassônica	40kHz

Fonte: (FLOP).

A Figura 10 apresenta a resposta do sensor à variação angular. Nota-se que o ângulo de visão do sensor é restrito, aproximadamente, a um arco de 30º.

Figura 10. Performance angular



Fonte: (WIKI).

3.4 SENSOR LIDAR

Capazes de estimar a distância dos objetos ao seu redor em duas ou três dimensões, utilizam tecnologia óptica, emitindo um feixe de luz laser que atinge um obstáculo e retorna ao sensor. A distância entre o sensor e o alvo é proporcional ao

tempo entre a emissão e recepção da luz pelo sensor. Os pulsos laser são emitidos sob uma taxa de frequência de repetição e realizam uma varredura perpendicular à direção de emissão. O sistema LIDAR utiliza ondas cujo comprimento varia entre 0,7µm e 1000µm, compreendendo a região do espectro eletromagnético denominada infravermelho próximo (WIKIPEDIA).

3.4.1 RP LIDAR A1

É um LIDAR 2D de baixo custo desenvolvido pela SLAMTEC, apresentado na

Figura *11*, que opera em 360º com 6 metros de alcance, de 5,5Hz a 10Hz numa velocidade de comunicação via USB de 115200bps. Os dados 2D obtidos por este sensor podem ser utilizados para mapeamento, localização e modelagem de um ambiente, *indoor* ou *outdoor* (SLAMTEC, 2016).

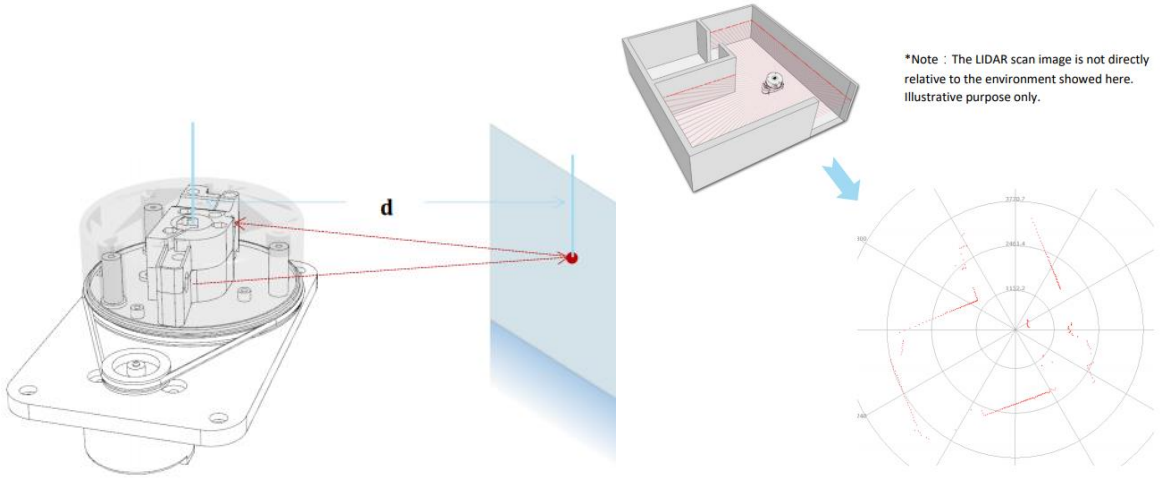
Figura 11. RP Lidar A1

Fonte: (SLAMTEC, 2016).

O usuário pode obter os dados de aquisição do sensor através da interface de comunicação serial/USB. Baseado no princípio de triangulação, o sistema mede a distância em mais de 2000 vezes por segundo com alta resolução de saída (<1% da distância) (SLAMTEC, 2016). O funcionamento do sensor pode ser observado na Figura 12, onde o sinal emitido pelo laser é refletido quando um objeto é detectado. O

DSP embarcado nos ensor processa os dados, retornando a distância e o ângulo entre o objeto e o RPLidar a partir da interface de comunicação.

Figura 12. Funcionamento do RPLidar A1



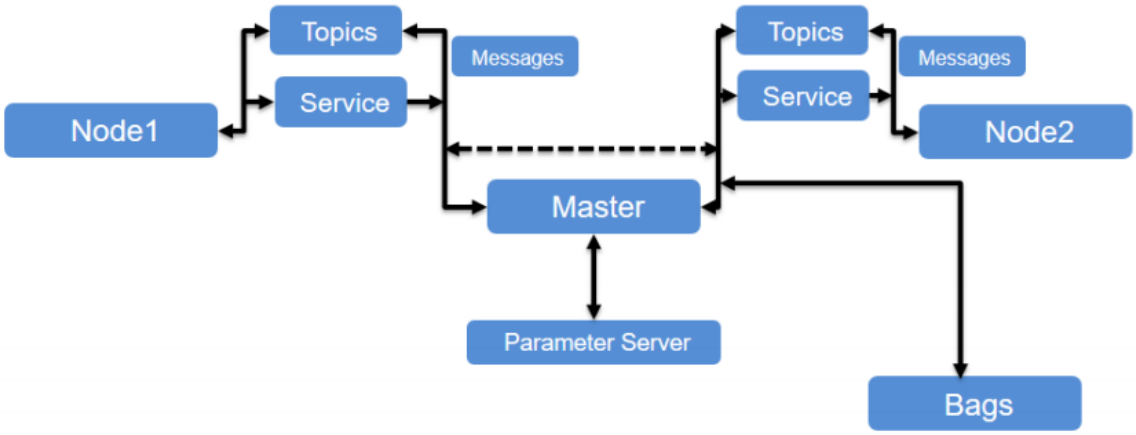
Fonte: (SLAMTEC, 2016).

3.5 ROBOT OPERATING SYSTEM (ROS)

Conjunto de ferramentas *open source* que fornecem serviços padrão do sistema operacional, como abstração de *hardware* e *drivers* de dispositivos, além de implementação de recursos incluindo detecção, reconhecimento, mapeamento, movimento, transmissão de mensagem entre processos, gerenciamento de pacotes, visualizadores e bibliotecas para desenvolvimento (ROS, 2014). A arquitetura ROS é dividida em três níveis de conceitos: *file system*, *computation graph* e *community*.

O primeiro nível consiste na estrutura interna necessária para o funcionamento do ROS, ou seja, como é realizada a organização de arquivos. Já no segundo nível ocorre a comunicação entre processos, na qual as aplicações se configuram como uma rede, permitindo a troca de mensagens. O fluxograma de funcionamento do nível *computation* pode ser visualizado na figura abaixo:

Figura 13 - Funcionamento ROS – Computation.



Fonte: (JOSEPH, 2017).

Conforme Figura 13, nota-se que há vários conceitos que devem ser compreendidos para o entendimento do ROS. Tais conceitos são detalhados a seguir:

- *Node*: os nós são processos; usualmente sistemas robóticos são constituídos por muitos nós;
- *Master*: o mestre opera como um nó intermediário que auxilia nas conexões entre diferentes nós. Dessa forma, o mestre tem informações de todos os nós em execução no ambiente do ROS, a partir das quais ocorre uma troca entre um nó e outro para o estabelecimento de uma conexão entre esses. A comunicação entre dois nós ROS inicia após a troca de informações.
- *Parameter Server*: o servidor de parâmetros é um repositório de informações onde são armazenados dados de configuração, disponibilizados aos nós para que esses os utilizem durante o tempo de execução. Um nó pode armazenar uma variável no servidor de parâmetros e definir sua privacidade.
- *Messages*: uma mensagem é uma simples estrutura de dados, a partir da qual os nós realizam a comunicação.
- *Topics*: método para troca de mensagens entre dois nós. As mensagens podem ser transportadas utilizando o padrão *publish-subscribe*, onde cada nó é um

executável que publica e subscreve em tópicos. Ao enviar dados a um tópico, diz-se publicar, e ao realizar a leitura de dados de um tópico, diz-se subscrever.

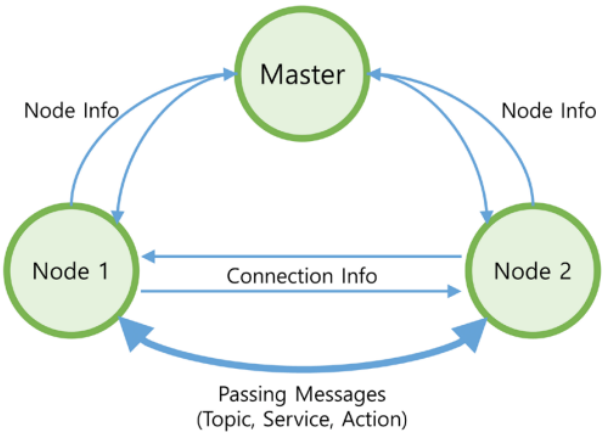
- *Services*: estabelece comunicação entre nós no modelo *request-reply*. Um serviço equivale a uma chamada de procedimento remoto, onde o nó cliente faz uma requisição e espera uma mensagem de resposta do nó servidor.
- *Bags*: utilitário para a gravação e reprodução de tópicos ROS. Possibilita gravar dados de um sensor e copiar para outros computadores, permitindo a reprodução desses.

O terceiro nível ROS compreende um conjunto de ferramentas e conceitos para compartilhar conhecimento, algoritmos e códigos entre desenvolvedores. Essencialmente, o ROS define uma infraestrutura de comunicação sobre a qual suas ferramentas e as aplicações desenvolvidas pelos usuários se estruturam. Em uma rede com máquinas rodando ROS, existirá sempre uma máquina mestre. O mestre inicializa todos os serviços necessários para que os nós ROS se comuniquem entre si. Um nó é basicamente um executável, sendo assim, é possível ter nós ROS rodando em qualquer uma das máquinas da rede. A comunicação entre os nós ocorre através do protocolo TCP/IP.

O mestre age como um servidor de nomes, pois armazena nomes de nós, tópicos, serviços e ações, o número da porta e os parâmetros. Na inicialização os nós registram suas próprias informações com o mestre e adquirem informações relativas de outros nós a partir do mestre. Em seguida, cada nó conecta-se diretamente entre si para executar a comunicação de mensagens. O fluxograma de comunicação pode ser visto na Figura 14.

As bibliotecas no sistema ROS são usadas para gravar nós ROS. Todos os conceitos são implementados em bibliotecas, ou seja, é possível utilizar ROS para uma determinada aplicação através de bibliotecas. As principais bibliotecas do ROS estão em C++ e Python.

Figura 14 - Fluxo de mensagens

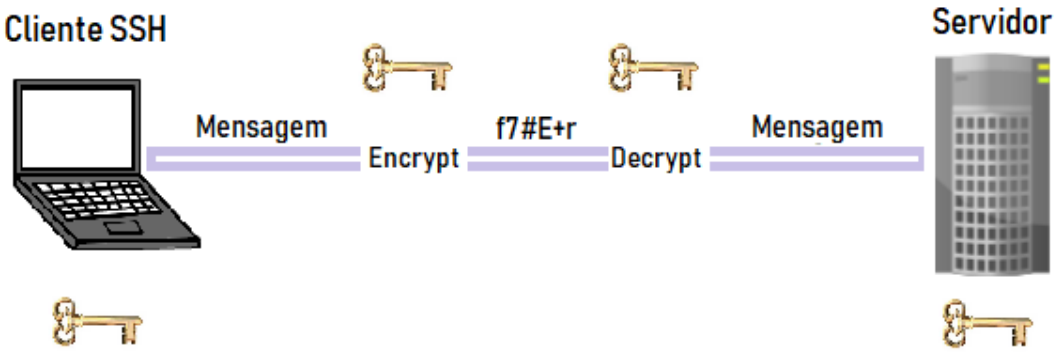


Fonte: (PYO, 2017).

3.6 SECURE SHELL

Conhecido como SSH, é um protocolo que permite a conexão segura à um servidor remoto pela internet. A conexão exige autenticação do servidor e do computador e é criptografada. Dessa forma, mesmo se alguém interceptar o pacote de dados que está sendo transmitido, não será possível visualizar o conteúdo da mensagem, pois apenas os computadores que estão conectados entre si possuem a chave para descriptografá-la, conforme Figura 15.

Figura 15. Conexão SSH



Fonte: (HOSTINGER).

Uma conexão SSH possibilita a execução de comandos remotamente, sendo equivalente à operação física do computador. Para iniciar o SSH é necessário indicar o usuário, correspondente à conta que se deseja acessar, além do endereço de IP da máquina que se deseja acessar. Por fim, basta apenas digitar a senha deste usuário. O acesso é iniciado com uma janela terminal remota.

Este protocolo é geralmente utilizado para execução de comandos em uma máquina remota, mas também suporta redirecionamento de portas TCP. Ainda, é possível transferir arquivos entre o cliente e o servidor usando os protocolos SSH *file transfer* (SFTP) ou *secure copy* (SCP).

4 VISÃO GERAL DO PROJETO

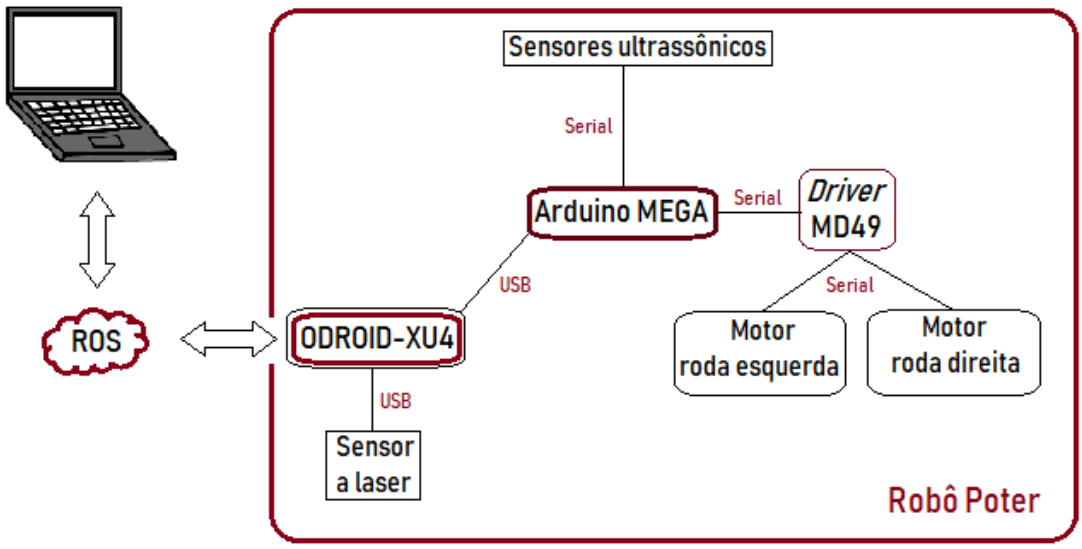
Com intuito de tornar o robô Poter uma plataforma móvel independente, o trabalho desenvolvido tem o dispositivo embarcado ODROID-XU4 como elemento central. A fim de simplificar o sistema de comunicação, o projeto utiliza ROS para integração dos componentes do Poter. Basicamente, a plataforma é composta por:

• Hardware: Arduino MEGA 2560, ODROID-XU4, motores;

• Software: ROS.

O Arduino MEGA é a responsável pela comunicação de nível baixo, recebendo sinais provenientes dos sensores ultrassônicos e enviando comandos ao driver do motor MD49. O microcontrolador realiza uma comunicação direta, via interface USB com o ODROID, o qual se comunica remotamente com o computador do usuário. Como realiza o processamento de todos os dados do sistema, o embarcado é o principal elemento do projeto. A utilização do ROS tem como principal vantagem a facilidade de integração dos componentes, visto que permite a utilização de bibliotecas já desenvolvidas pela comunidade, abstraindo a comunicação de baixo nível da plataforma. A arquitetura do projeto pode ser observada na Figura 16.

Figura 16. Visão geral do sistema



Fonte: (adaptado de SCHEIK)

5 METODOLOGIA

5.1 ROBÔ MÓVEL

Apresentado na Figura 17, o robô móvel utilizado neste trabalho foi desenvolvido no GARRA, da UFSM. É constituído mecanicamente por uma base de madeira, duas rodas dianteiras, além de uma terceira roda para auxiliar na estabilidade do sistema. Utiliza dois motores de corrente contínua modelo EMG49 nas rodas dianteiras, os quais tem como tarefa suportar a estrutura mecânica e realizar tração. A Figura 18 ilustra o modelo dos motores utilizados.

Utiliza uma base de madeira MDF com 18mm de espessura, a fim de suportar o esforço mecânico de sensores e sistemas que envolvam notebook. Sobre a base do robô, no interior de um *case*, estão localizados o microcontrolador Arduino MEGA e o driver de controle dos motores, conforme pode-se visualizar na figura abaixo:

Figura 17. Robô Poter

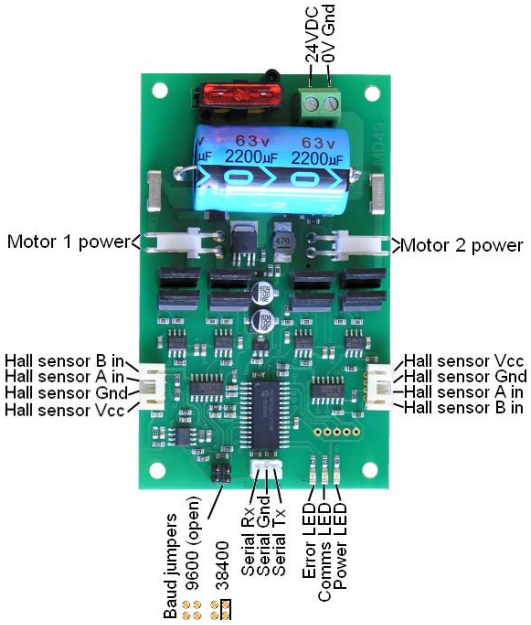
Fonte: (NASCIMENTO, 2017).

Aos motores estão acoplados encoders incrementais que possibilitam o uso de odometria, bem como o driver MD49 (Figura 19), responsável pela leitura dos encoders, controle e proteção de tais elementos. O driver é capaz de realizar o controle dos dois motores de forma combinada ou independente, com aceleração variável e potência regulável. Possui proteção contra curto-circuito, sobrecorrente e sobretensão e opera via comunicação serial de 9600bauds ou 3400bauds. A Tabela 4 apresenta as especificações dos motores utilizados no robô.

Figura 18 - Motor EMG49

Fonte: (ELECTRONICS).

Figura 19 - Driver MD49



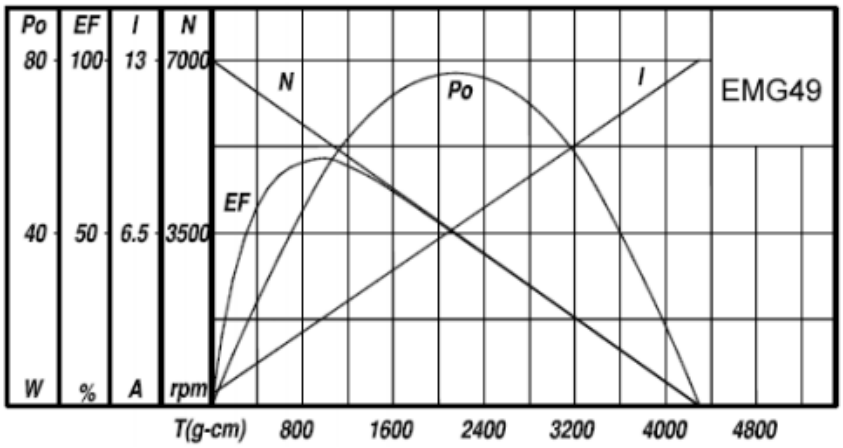
Fonte: (ELECTRONICS).

Tabela 4 - Especificações dos motores

Especificações EMG49	
Tensão nominal	$24V_{DC}$
Torque nominal	16kg/cm
Velocidade nominal	122RPM
Corrente nominal	2100mA
Velocidade sem carga	143RPM
Corrente sem carga	500mA
Corrente em torque máximo	13A
Potência nominal de saída	34,7W
Contagem do encoder por volta do eixo	980

Fonte: (ELECTRONICS).

Figura 20. Características do motor



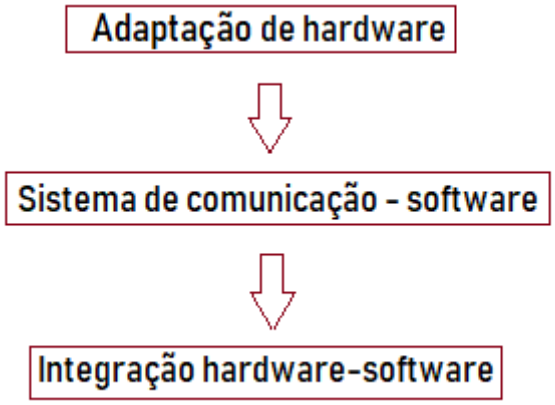
Fonte: (ELECTRONICS).

Na configuração padrão do *driver*, a faixa de velocidade de operação dos motores é de 0 a 255, sendo 0 a velocidade máxima reversa, 128 velocidade nula e 255 velocidade máxima para frente, a qual é setada pelo usuário via interface serial.

Para alimentação dos motores é utilizada uma fonte externa, de 30V. Ainda, visto que o sistema inicial não apresenta um computador embarcado, o uso de *notebook* sobre a base do robô se faz necessário para programação da plataforma

robótica, alimentação do Arduino e realização de testes. O trabalho realizado na plataforma robótica foi desenvolvido em etapas, conforme Figura 21. As etapas serão detalhadas nas seções a seguir.

Figura 21. Etapas do projeto



Fonte: (Autora do trabalho).

5.2 ADAPTAÇÃO DE HARDWARE

Foram realizadas algumas modificações na base do veículo robótico, a fim de adaptá-lo à integração sensorial e comunicação com o dispositivo embarcado. Inicialmente implementou-se os sensores ultrassônicos descritos em 3.3.1 , os quais são responsáveis pela detecção de obstáculos pelo robô. Para tanto, o estudo inicial focou na melhor disposição de tais dispositivos na plataforma, a partir da realização de testes de viabilidade de uso dada as respostas experimentais destes sensores ao ambiente.

5.2.1 Testes práticos – HC-SR04

Inicialmente testou-se a precisão destes dispositivos quanto à medição de distância. No teste realizado o sensor foi colocado a diferentes distâncias, aferidas com uma trena manual, em relação à parede de uma sala. Foram realizadas várias medições, com dois sensores. Notou-se que o sensor é capaz de detectar obstáculos

a partir de 1,5cm abaixo de seu eixo central. A resposta obtida corresponde ao código teste apresentado no ANEXO A, e pode ser observada na Tabela 5.

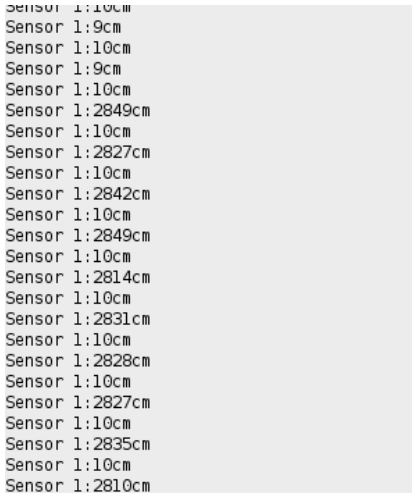
Tabela 5 - Medições de distância - sensor ultrassônico

Distância trena (cm)	***Distância sensor (cm)***	***Erro (cm)***
5	5	0
10	10	0
15	15	0
30	31	1
50	51	1
80	82	2
100	103	3
200	202	2
250	253	3
300	310	10
350	360	10
370	383	13
400	2830	2430

Fonte: (Autora do trabalho).

A fim de verificar a viabilidade deste sensor no ambiente, o mesmo foi exposto à obstáculos de diferentes materiais: madeira, plástico, tecido, metal, papelão; para os quais retornou respostas ótimas. Entretanto, quando exposto à espuma, o sensor apresentou grandes variações, conforme Figura 22.

Figura 22. Resposta do sensor a um objeto de espuma



Fonte: (Autora do trabalho).

Ainda, como variável a ser considerada, tem-se o tempo de respostas dos sensores. Visto que o robô deve atuar de forma eficiente à detecção de obstáculos, justifica-se a preocupação ao tempo de resposta dos mesmos. Conforme (SANTOS, 2015), supondo um objeto à distância máxima teórica de detecção do sensor, 4m de distância, estimou-se o tempo de resposta do mesmo até o envio de comando aos motores. Cada sensor exige 10µs de *trigger*, considerando a velocidade do som no ar, bem como o fato de que tempo de resposta equivale ao dobro da distância lida, tem-se:

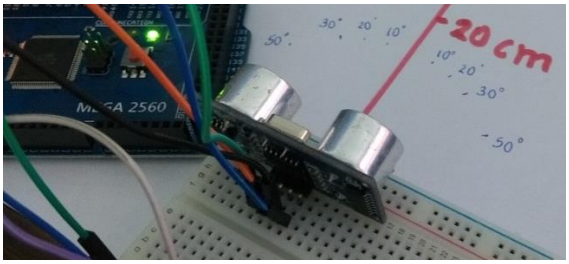
$$Tempo\ de\ resposta\ de\ cada\ sensor: 10\mu s + {}^{8m}/_{340m/s} = \ 23{,}54\mu s$$

Como o *clock* do Arduino é de 16MHz e a frequência de leitura dos sensores para a máxima distância é de 42,48Hz, o tempo de processamento é desprezível, e pode-se afirmar que o veículo é capaz de responder a tempo aos obstáculos do ambiente.

Por fim, foram feitos testes práticos a fim de corroborar o melhor ângulo de operação dos sensores, o qual se espera resultados semelhantes à Figura 10. Para tanto, um sensor foi colocado frente a uma parede, e o ângulo de medição do mesmo foi variado em relação ao eixo central do sensor a diferentes distâncias, a fim de

verificar a capacidade de detecção do mesmo dada às variações angulares. A variação angular foi aferida com auxílio de um transferidor, conforme pode ser observado na Figura 23.

Figura 23. Teste de medição sob variações angulares



Fonte: (Autora do trabalho).

Os resultados obtidos podem ser analisados na tabela abaixo. Nota-se que quanto maior a distância e a variação angular, maior o erro obtido na leitura.

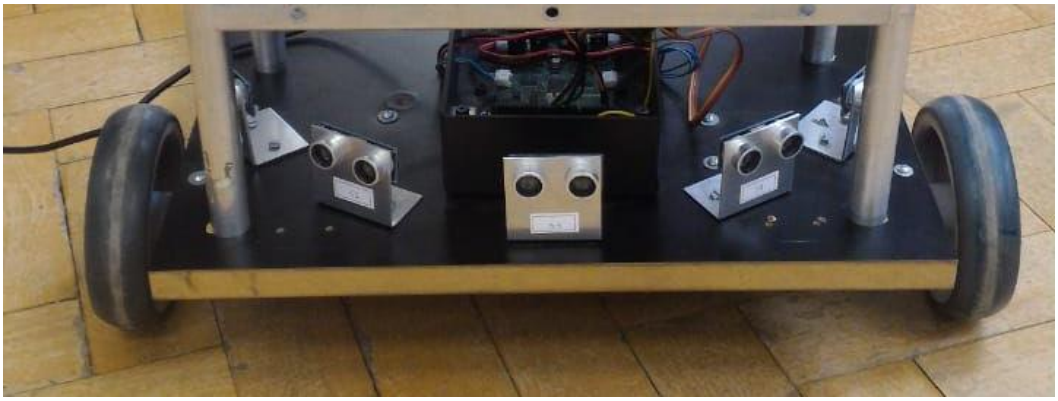
Tabela 6 - Resultados obtidos para variação angular

Distância trena (cm)	***Ângulo (º)***	***Média (cm)***	***Erro (cm)***
60	17	65	5
	10	62	2
	0	60	0
90	12	95	5
	10	91	1
	0	90	0
120	22	111	9
	15	122	2
	0	118	2
150	25	168	18
	10	154	4
	0	148	2

Fonte: (Autora do trabalho).

Sendo assim, fez-se a alocação de 7 sensores no robô: três dispostos à frente, dois na diagonal dianteira e dois na diagonal traseira, conforme Figura 24. A partir dos resultados obtidos, considerando que cada um dos 5 sensores frontais possui um arco total de leitura de 30º, é possível estimar que em 180º à frente do robô haverá detecção de aproximadamente 80% dos obstáculos em uma distância máxima de 370cm, de forma que o mesmo possa explorar um ambiente desviando objetos de maneira satisfatória.

Figura 24. Disposição dos sensores no veículo robótico



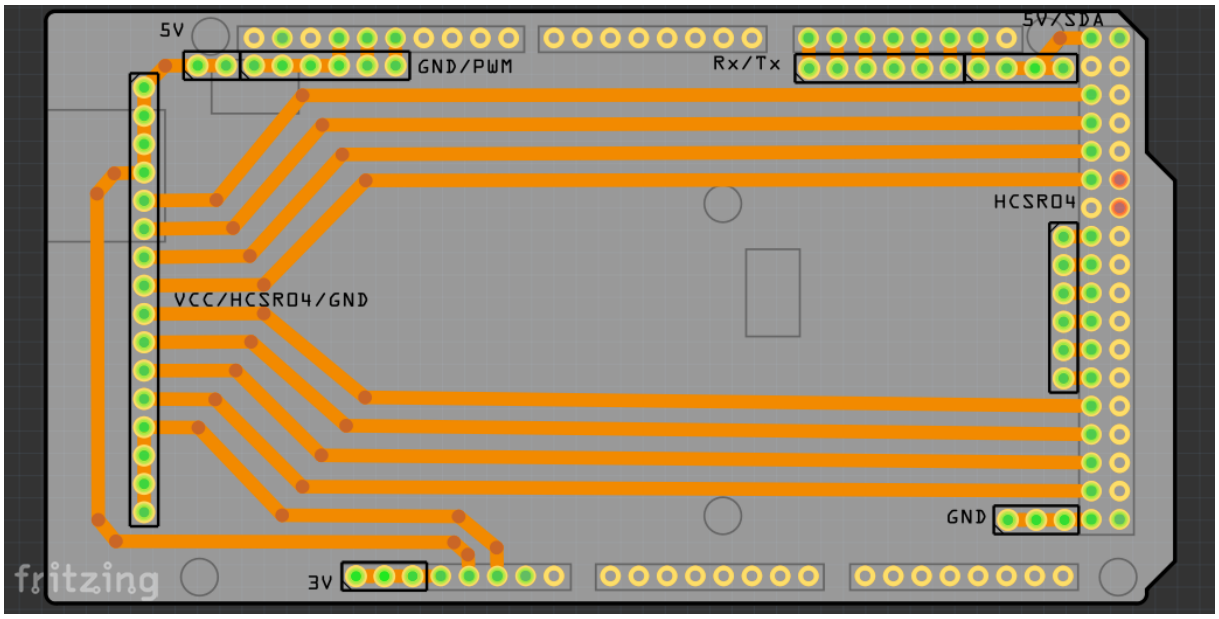
Fonte: (Autora do trabalho).

5.2.2 Placa de circuito impresso

A partir da escolha de implementar 7 sensores ultrassônicos, percebeu-se a necessidade de fabricação de uma placa de circuito impresso, de forma a possibilitar a utilização desses em conjunto, conectados à um Arduino MEGA. Visto que o microcontrolador já estava sendo utilizado para enviar comandos aos motores, como proposta de solução fez-se um *shield*, a fim de facilitar o uso de ambos simultaneamente.

Projetou-se o *shield* no software Fritizing, priorizando a alocação de trilhas em apenas um lado da placa, para facilitar a confecção manual da mesma, bem como a disposição destas em curvas suaves, de 45º, a fim de evitar a ocorrência de interferências eletromagnéticas externas. O projeto pode ser visualizado na Figura 25.

Figura 25. Placa projetada

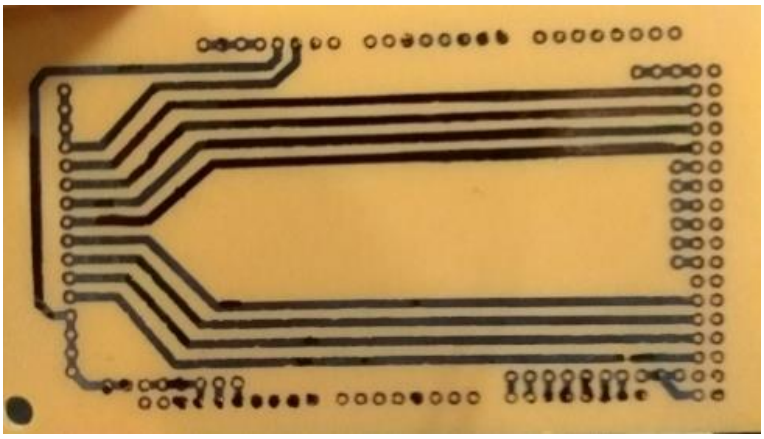


Fonte: (Autora do trabalho).

A placa projetada foi impressa à laser nas cores preto e branco em um papel *glossy* de gramatura 180. A partir da impressão utilizou-se de propriedade térmica para transferir o desenho das trilhas e ilhas para uma placa de fenolite virgem.

O processo de confecção manual consistiu, primeiramente, na limpeza da placa com palha de aço e álcool, a fim de retirar quaisquer camadas de gordura presentes e facilitar a fixação da tinta na mesma. Posteriormente, posicionando-se a folha impressa sobre a placa, aplicou-se calor à mesma, pressionando-a durante 10 minutos, de forma que as trilhas e ilhas fossem transferidas totalmente à placa. Após a transferência, a placa foi submersa em percloreto de ferro, por aproximadamente 25 minutos, a fim de corroer todo o cobre não coberto pela tinta da impressão. O resultado inicial obtido nesse processo pode ser visto na Figura 26.

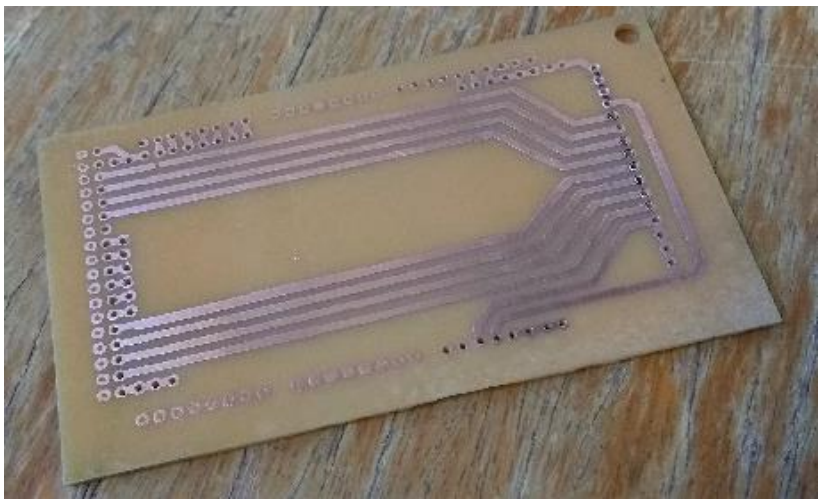
Figura 26. Placa após transferência das trilhas e ilhas



Fonte: (Autora do trabalho).

Nota-se que o cobre permaneceu apenas onde estava coberto pela tinta, a qual protegeu o cobre da corrosão. Posteriormente, fez-se a perfuração das ilhas, bem como a limpeza da placa, com auxílio de palha de aço (Figura 27). Por fim, aplicou-se às trilhas uma solução de breu, a fim de protegê-las contra oxidação.

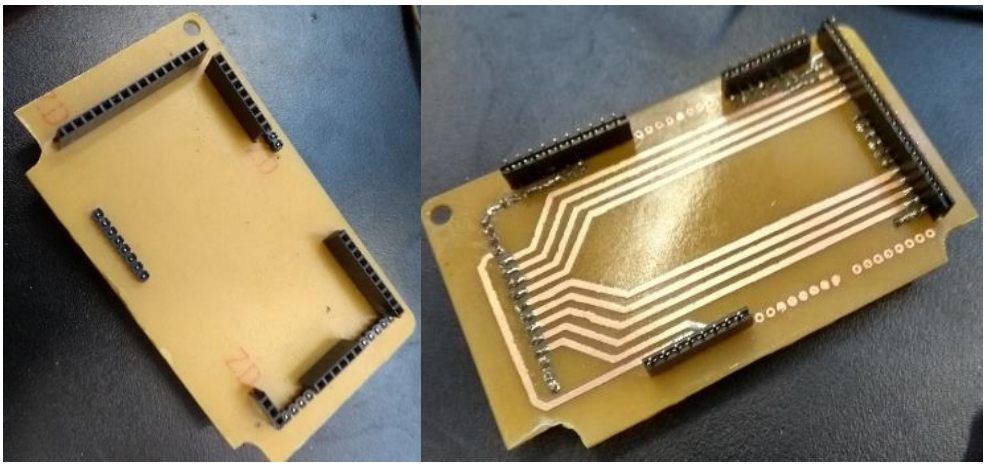
Figura 27. Placa após limpeza



Fonte: (Autora do trabalho).

Ainda, para confeccionar o *shield*, fez-se a soldagem dos *headers* fêmea e macho, conforme projeto, com estanho, resultado na Figura 28.

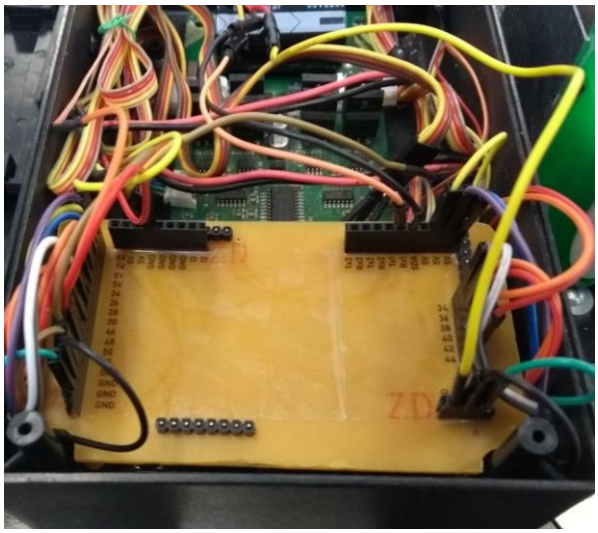
Figura 28. Placa após conexão dos *headers*



Fonte: (Autora do trabalho).

As conexões da placa, foram testadas com multímetro, a fim de evitar possíveis erros durante sua utilização. A identificação dos pinos da placa foi realizada por meio de adesivo. O *shield* finalizado, conectado ao Arduino do robô, pode ser visto na Figura 29.

Figura 29. Placa finalizada

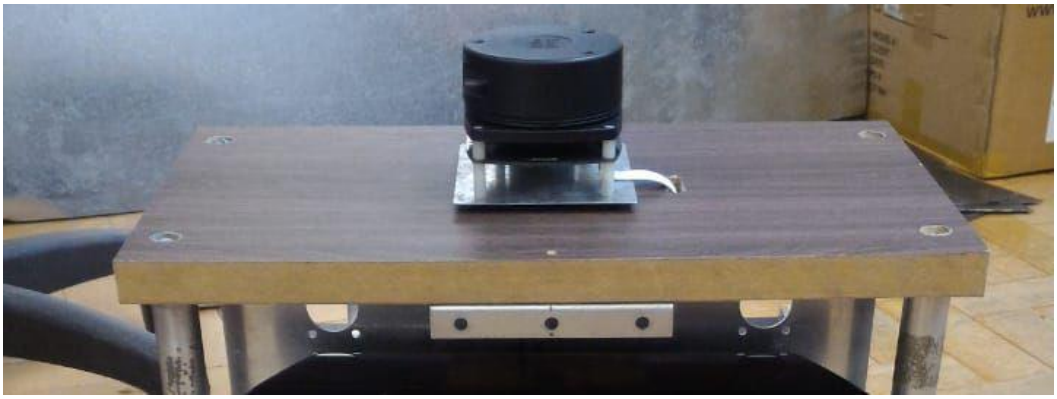


Fonte: (Autora do trabalho).

5.2.3 RPLidar

Dado o modo de funcionamento do RPLidar, como parâmetro para implementação deste sensor no veículo robótico percebeu-se a necessidade de que o mesmo ficasse no nível mais alto da plataforma, de forma a mapear todo o ambiente em 360º sem coincidir com obstáculos do próprio corpo do robô. Assim, o RPLidar foi alocado na plataforma superior, a partir de uma base de alumínio conectada à base de madeira por meio de um velcro, conforme Figura 30.

Figura 30. Posicionamento do RPLidar no Poter



Fonte: (Autora do trabalho).

5.2.4 Odroid

O suporte para o dispositivo embarcado foi implementado de forma a possibilitar a fixação do mesmo à base do robô, permitir o acesso às portas USB e pino de alimentação deste dispositivo, além de facilitar a dissipação de calor, conforme Figura 31.

Figura 31. Odroid acoplado ao Poter

Fonte: (Autora do trabalho).

5.2.5 Sistema de alimentação

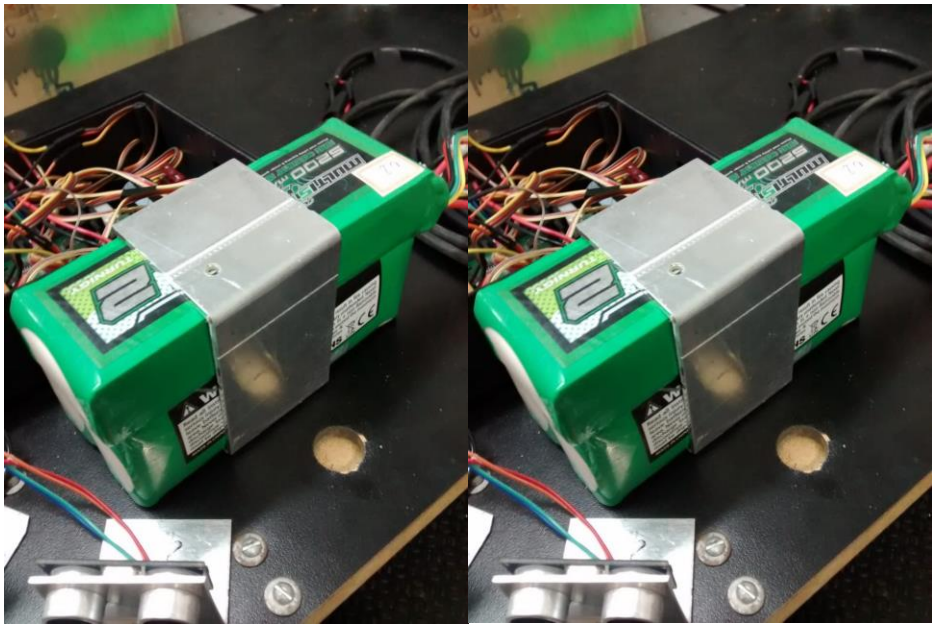
Inicialmente, o sistema de alimentação do robô era composto por uma fonte de tensão de 30V, configurada em 23V para alimentação dos motores, além de um computador para programar e alimentar o microcontrolador via USB. Como alternativa, a fim de possibilitar a alimentação independente do robô, tornando-o um sistema embarcado, fez-se algumas alterações na base do veículo, as quais serão descritas ao longo desta seção.

Propôs-se a realização da alimentação dos motores com duas baterias de 14,8V conectadas em série, com ajuste de tensão a partir de um regulador ajustável. As baterias utilizadas são da marca Multistar, de 5200mAh com 4 células e permitem o uso de dispositivo para balanceamento de carga. O regulador utilizado para esta conexão é um *step down,* LM2596, que trabalha com 4V a 40V na entrada e fornece de 1,5V a 37V na saída. Ajustou-se a saída do regulador em 23V, conectando-a ao *driver* dos motores.

Para alimentação do Odroid, utiliza-se uma bateria semelhante àquelas citadas anteriormente em conjunto de um regulador de tensão *step down*, que suporta até

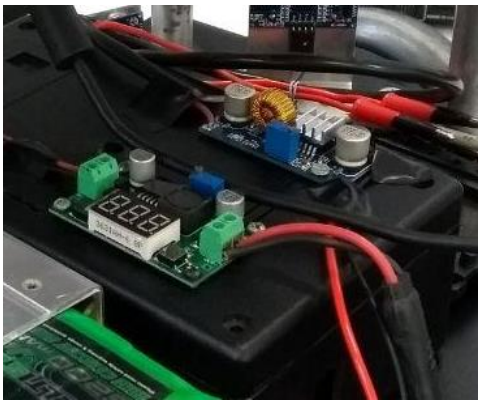
30V na entrada e fornece até 1,5V na saída. Ajustou-se a tensão de saída do regulador em 4,8V conectando-a ao embarcado. A adaptação realizada pode ser observada nas Figura 32 e 33.

Figura 32. Adaptação das baterias ao robô



Fonte: (Autora do trabalho).

Figura 33. Reguladores de tensão utilizados



Fonte: (Autora do trabalho).

5.3 ADAPTAÇÃO DE SOFTWARE

O controle do robô é o módulo principal do sistema, tendo por objetivo permitir acesso e monitoramento à distância. Assim sendo, a finalidade desta seção é detalhar o desenvolvimento da interface de comunicação entre robô e usuário, possibilitando o controle de movimentos do robô para frente, trás, direita e esquerda.

O estudo inicial consistiu na comunicação entre o dispositivo Odroid e um computador pessoal, o qual seria responsável pelo controle à distância do veículo robótico.

5.3.1 Implementação da rede de comunicação

A rede de comunicação da plataforma foi implementada a partir do ROS, dada a facilidade de desenvolvimento de integração de dispositivos e interface de controle neste subsistema operacional.

Primeiramente, foi necessário realizar a instalação e configuração do ROS no Odroid. ROS funciona sob sistemas operacionais Linux, portanto, as máquinas a serem conectadas ao robô devem possuir alguma distribuição do Linux, bem como os pacotes ROS instalados. O Ubuntu 16.04 foi instalado no eMMC do Odroid. Foi necessário realizar o *download* da distribuição em um computador e transferi-la ao eMMC por meio do *software* Etcher e adaptadores eMMC–SD e SD–USB. Ao conectar o eMMC ao Odroid, fez-se a configuração do Ubuntu normalmente. A tela de login após a instalação pode ser vista na Figura 34.

Realizou-se a instalação da distribuição ROS Kinetic Kame, o qual possui grande acervo de bibliotecas estáveis e suporte ao Ubuntu 16.04, além de apresentar bom funcionamento em processadores ARM, semelhantes ao Odroid. A configuração foi realizada manualmente via terminal. Os requisitos necessários para instalação podem ser vistos em (ROS).

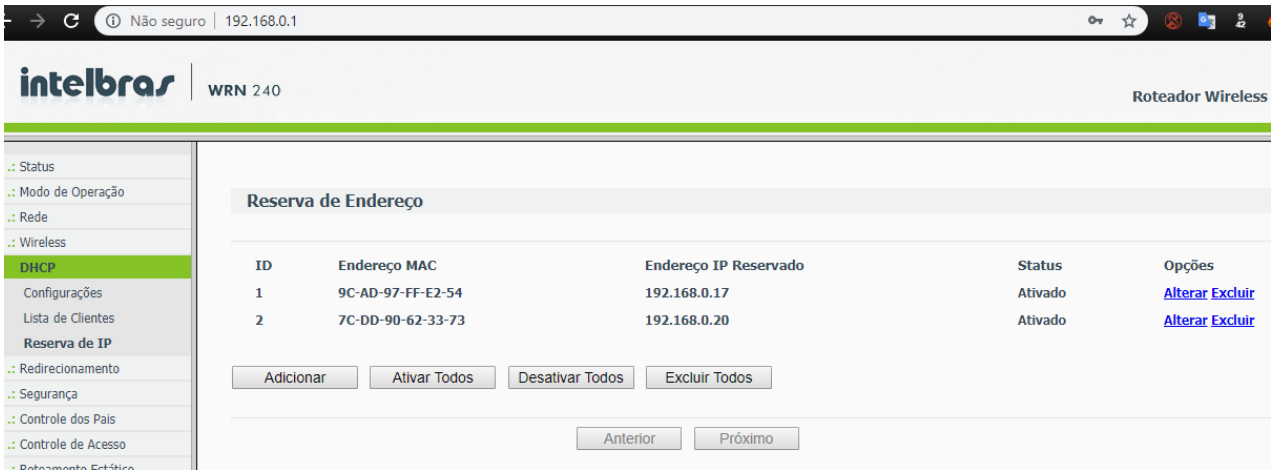
De forma a viabilizar a comunicação do computador pessoal via ROS com o dispositivo embarcado, foi necessário instalar a mesma distribuição ROS a esta máquina. A configuração do subsistema foi realizada via terminal, onde refez-se os passos de instalação anteriormente realizados no Odroid.

Figura 34. Tela de login no dispositivo embarcado

Fonte: (Autora do trabalho).

Após a instalação do ROS nas máquinas a serem conectadas ao robô, fez-se uma rede de comunicação entre ambas. Utilizou-se um roteador para desenvolver uma rede local entre os dispositivos. No endereço base do roteador, configurou-se a rede na qual os dispositivos foram vinculados à IPs estáticos, bem como determinou-se o endereçamento de portas. Assim, a rede ROS poderia iniciar a comunicação automaticamente aos IPs vinculados sem reconfigurá-los toda a vez que algum dispositivo fosse reiniciado. A configuração do roteador pode ser vista na Figura 36.

Figura 35. Configuração do roteador

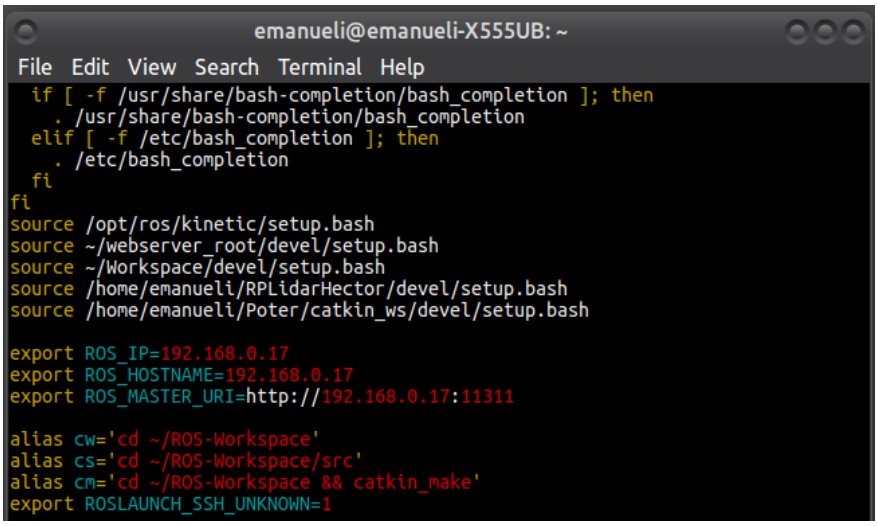


Fonte: (Autora do trabalho).

A rede foi configurada no endereço 192.168.0.1, SSID Poter, sendo protegida por senha.

O computador pessoal foi configurado na rede ROS como dispositivo mestre, vinculando-o ao endereço de IP. Assim, esse é o dispositivo principal da rede, capaz de iniciar a comunicação com o robô, pode ser observada na Figura 36.

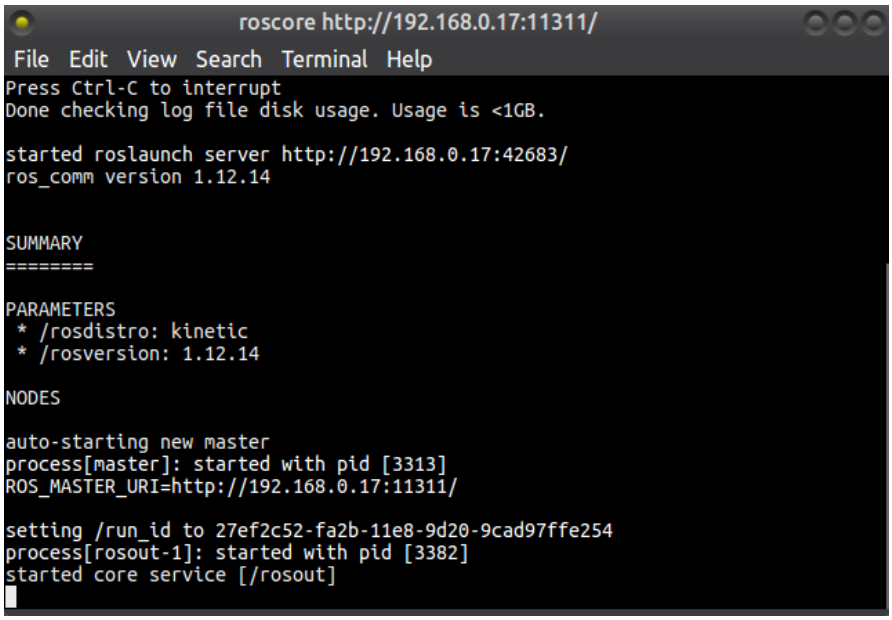
Figura 36. Configuração dispositivo mestre



Fonte: (Autora do trabalho).

Para iniciar o mestre, é necessário dar o comando *roscore*, o qual retorna uma resposta correspondente à Figura 37, caso não haja nenhum erro na rede e a configuração tenha sido realizada corretamente:

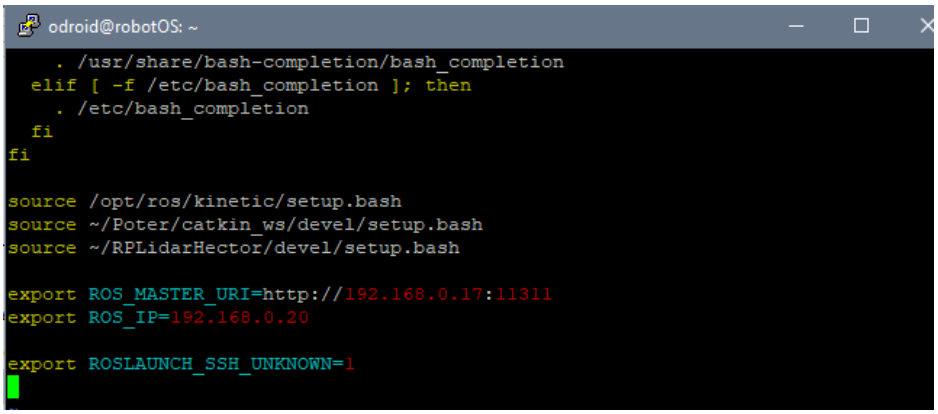
Figura 37. Iniciando o mestre



Fonte: (Autora do trabalho).

Nota-se que o comando retorna o endereço da máquina, anteriormente configurado, bem como a versão da distribuição ROS. Já o Odroid, foi configurado como cliente (Figura 38), vinculado a seu endereço de IP. Um cliente é dependente do dispositivo mestre, apenas podendo iniciar um processo quando o mestre estiver na rede:

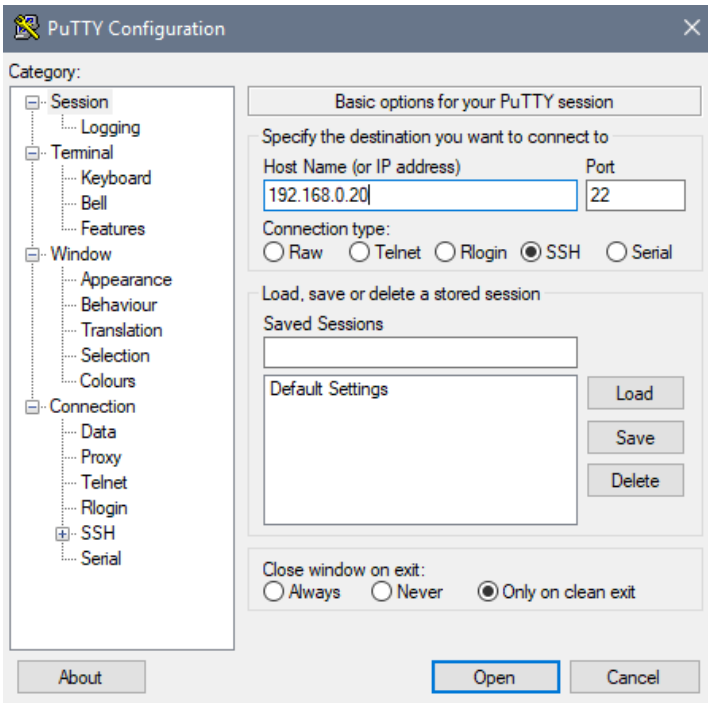
Figura 38. Configuração dispositivo cliente



Fonte: (Autora do trabalho).

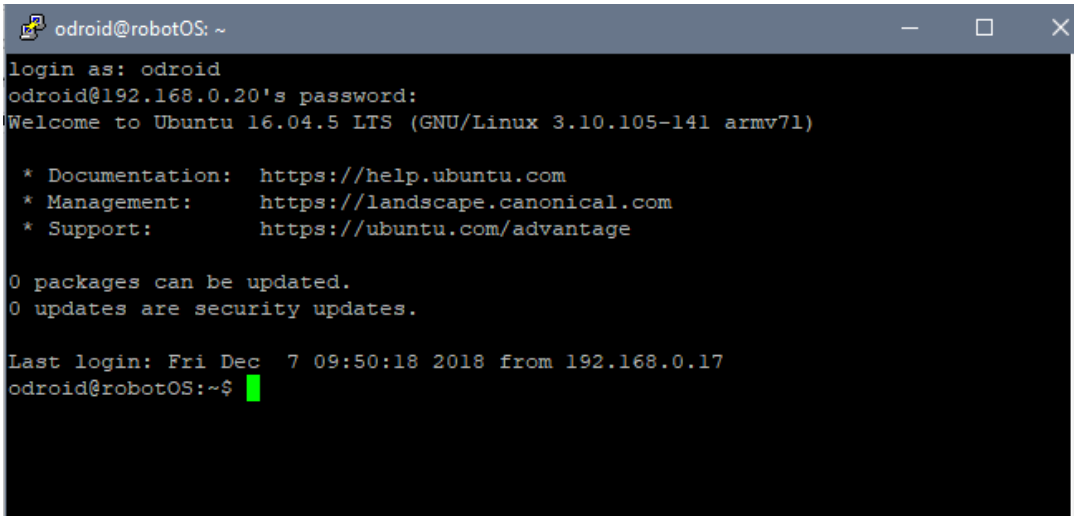
A execução de comandos no Odroid pode ser realizada via protocolo SSH, o qual pode ser iniciado facilmente via terminal em computadores com Ubuntu, ou por meio do software *Putty* em computadores com Windows (Figuras 39 e 40).

Figura 39. Tela de acesso no software Putty



Fonte: (Autora do trabalho).

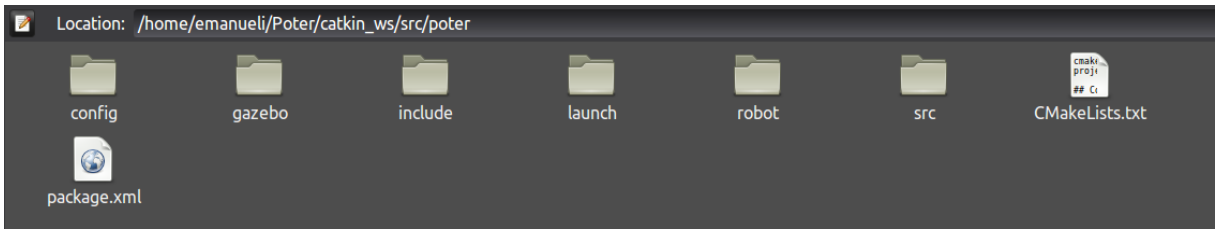
Figura 40. Acesso permitido via software Putty



Fonte: (Autora do trabalho).

O trabalho desenvolvido em ROS se baseia, principalmente, em pacotes. Um pacote é uma pasta de arquivos, que contém todas as informações necessárias para realizar uma tarefa. A pasta principal de um pacote é chamada *workspace*, visto que a partir desta são realizados os comandos para compilar todos os arquivos presentes no pacote. A imagem abaixo mostra a *workspace* criada para este projeto. A pasta *src* contém os códigos fonte do projeto.

Figura 41. Workspace Poter

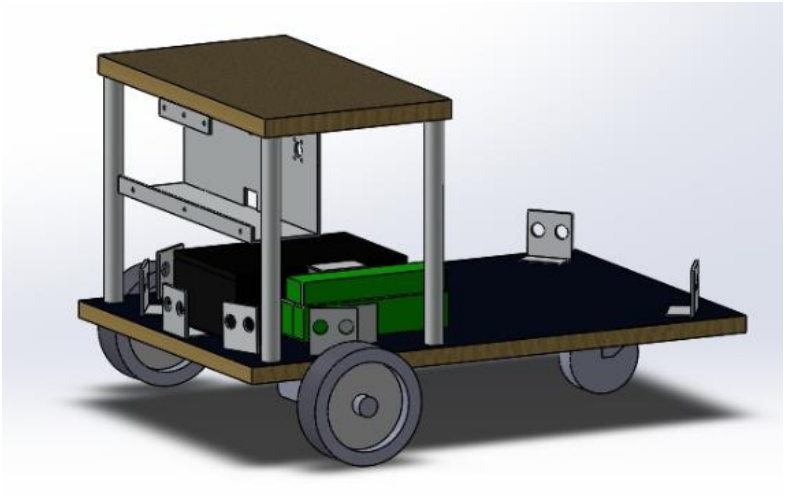


Fonte: (Autora do trabalho).

5.3.2 Modelagem 3D do robô

ROS possui suporte a simulações do comportamento de dispositivos robóticos em ambientes. As simulações possuem suporte gráfico e simulação de elementos 3D. A fim de explorar tais ferramentas e facilitar a realização de trabalhos futuros nos ambientes de simulação ROS pelo grupo GARRA, fez-se uma modelagem 3D do veículo robótico. Cada peça do robô foi projetada separadamente, sendo por fim realizada uma montagem completa do mesmo. Foram consideradas as medidas reais, bem como o grau de liberdade de movimento de cada peça. As rodas dianteiras foram projetadas com movimento rotativo limitado à um único eixo, enquanto a roda traseira foi projetada para girar em torno de seu próprio eixo. O projeto finalizado pode ser visualizado na Figura 42.

Figura 42. Modelagem 3D do robô



Fonte: (Autora do trabalho).

A partir de um *plugin*, exportou-se o modelo projetado para um formato reconhecido pelo ROS. Tal formato é o URDF, *Unified Robot Description Format*, o qual descreve um robô, suas peças e dimensões, bem como a manipulação do mesmo nas simulações dado o grau de liberdade de movimento de cada peça.

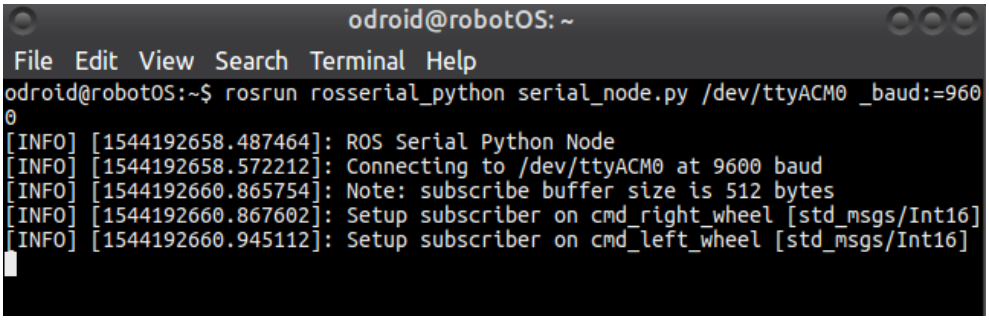
5.3.3 Interface ROS - Arduino

O Arduino acoplado ao robô interage com o ambiente externo através de sensores, que captam informações e as enviam ao microcontrolador; e atuadores, que são acionados após o processamento destas informações pelo Arduino. Os atuadores podem realizar tarefas pré-programadas ou acionadas em tempo real por um usuário com acesso à rede local.

A fim de possibilitar o interfaceamento do ROS com o Arduino, fez-se a instalação do pacote *rosserial* no Odroid e no dispositivo mestre, o qual facilita o uso de diferentes dispositivos controlados via comunicação serial. O repositório do pacote está disponível no *Github*, sendo necessário apenas cloná-lo a um ambiente de trabalho ROS.

Ainda, foi necessário instalar a biblioteca *ros_lib* no Arduino, para permitir a comunicação via ROS. Por fim, para iniciar um nó capaz de subscrever e publicar tópicos no Arduino, basta enviar um comando via terminal especificando a porta em que o dispositivo está conectado e a velocidade de comunicação serial configurada como mostrado na Figura 43.

Figura 43. Iniciando comunicação com o Arduino



Fonte: (Autora do trabalho).

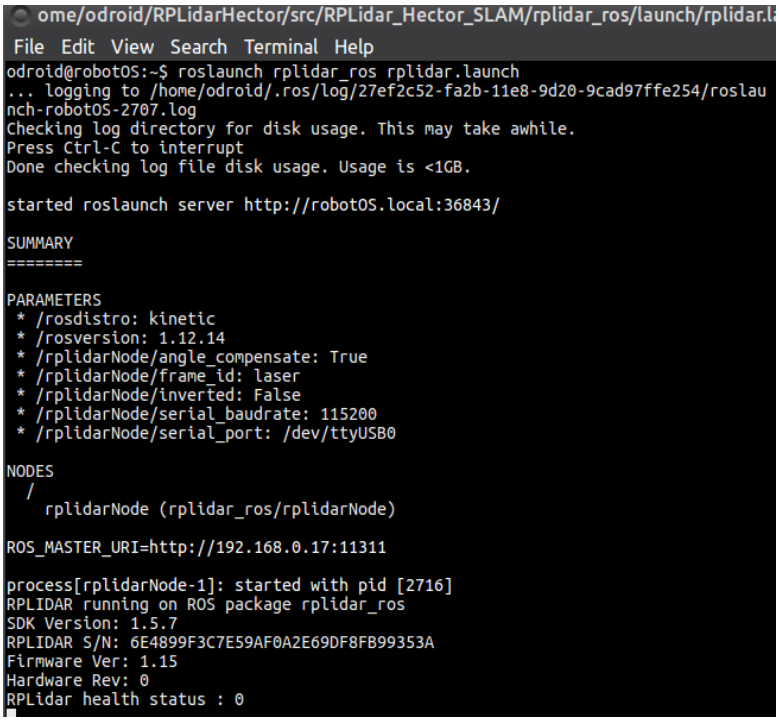
A partir desse comando, o usuário está habilitado para se comunicar com o Arduino, facilitando o acesso e controle de sensores e atuadores no robô. Automaticamente, a comunicação estabelecida identifica as chamadas ROS declaradas no código presente no microcontrolador, bem como o tipo da mensagem.

Na imagem apresentada acima, tem-se que é possível subscrever os tópicos *cmd_right_wheel* e *cmd_left_wheel*, os quais recebem mensagens do tipo inteiro de 16bits.

5.3.4 Integração RPLidar - ROS

A comunidade ROS disponibiliza um pacote para o sensor RPLidar, o qual abstrai os dados enviados pelo sensor via USB publicando-os em tópicos, os quais podem ser processados e validados pelo usuário. Após o comando para iniciar o pacote, o console retorna parâmetros como a porta em que está conectado o sensor e velocidade de comunicação, conforme Figura 44.

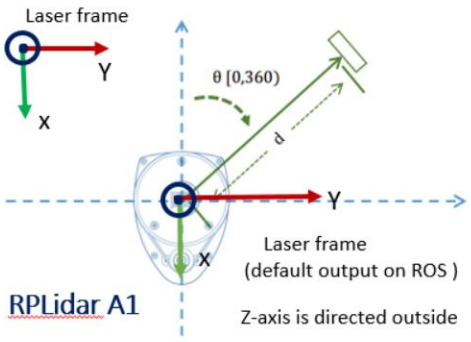
Figura 44. Iniciando a comunicação com o RPLidar



Fonte: (Autora do trabalho).

Na Figura 45 observa-se o posicionamento no qual o sensor efetua as medições. A medição angular é realizada ao longo da coordenada Z, no sentido horário:

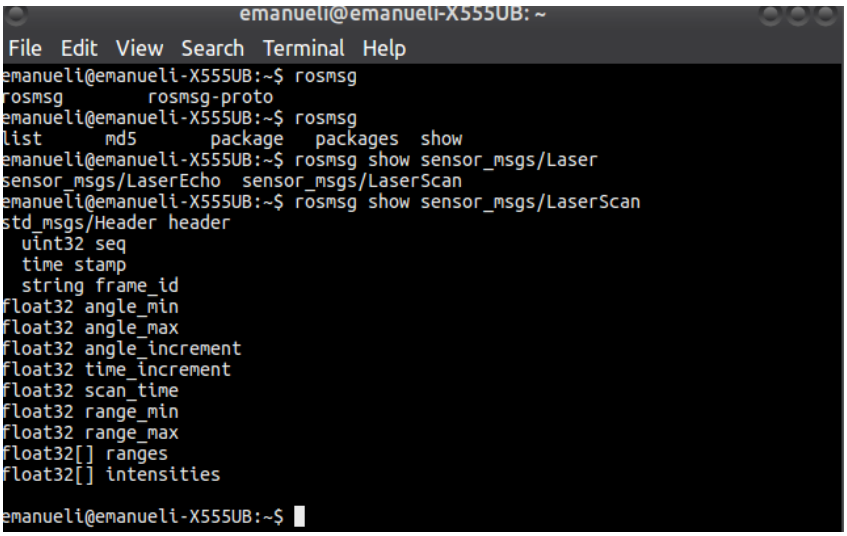
Figura 45. Posicionamento RPLidar



Fonte: (ROBOPEAK).

Os dados enviados pelo sensor são do tipo *sensor_msgs*, *LaserScan*, o qual retorna informações relativas às medições realizadas. Na Figura 46 observa-se os parâmetros retornados por este tipo de mensagem. Tem-se a medição angular máxima e mínima, em radianos, bem como o intervalo angular entre cada medição. As informações em *range_max* e *range_min* correspondem, respectivamente ao alcance máximo e mínimo que o sensor pode retornar, em metros. *Ranges* correspondem à todas as medições realizadas durante o escaneamento.

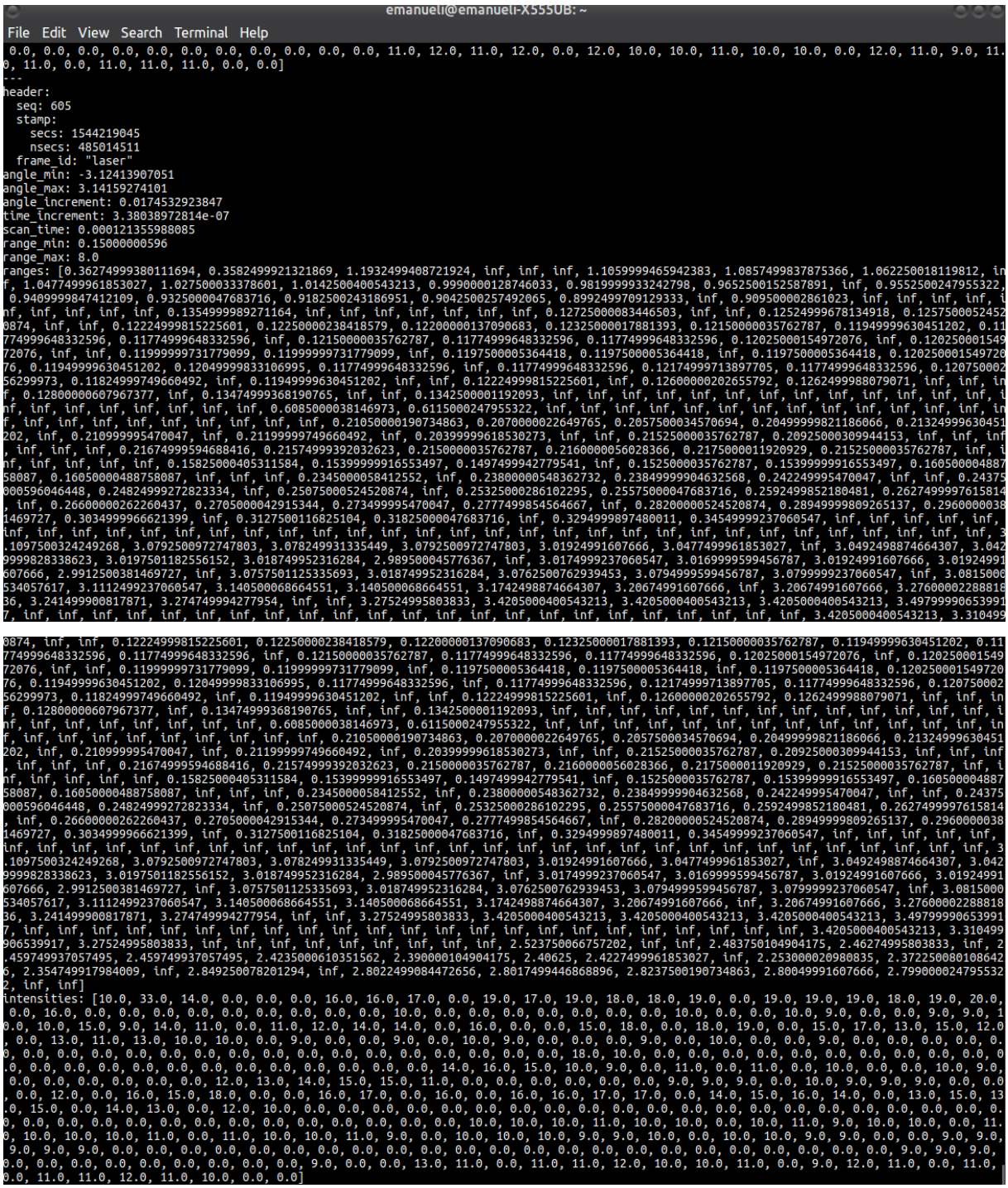
Figura 46. Parâmetros de mensagem do sensor



Fonte: (Autora do trabalho).

A Figura 47 mostra a leitura das mensagens *LaserScan* publicadas no tópico */scan*.

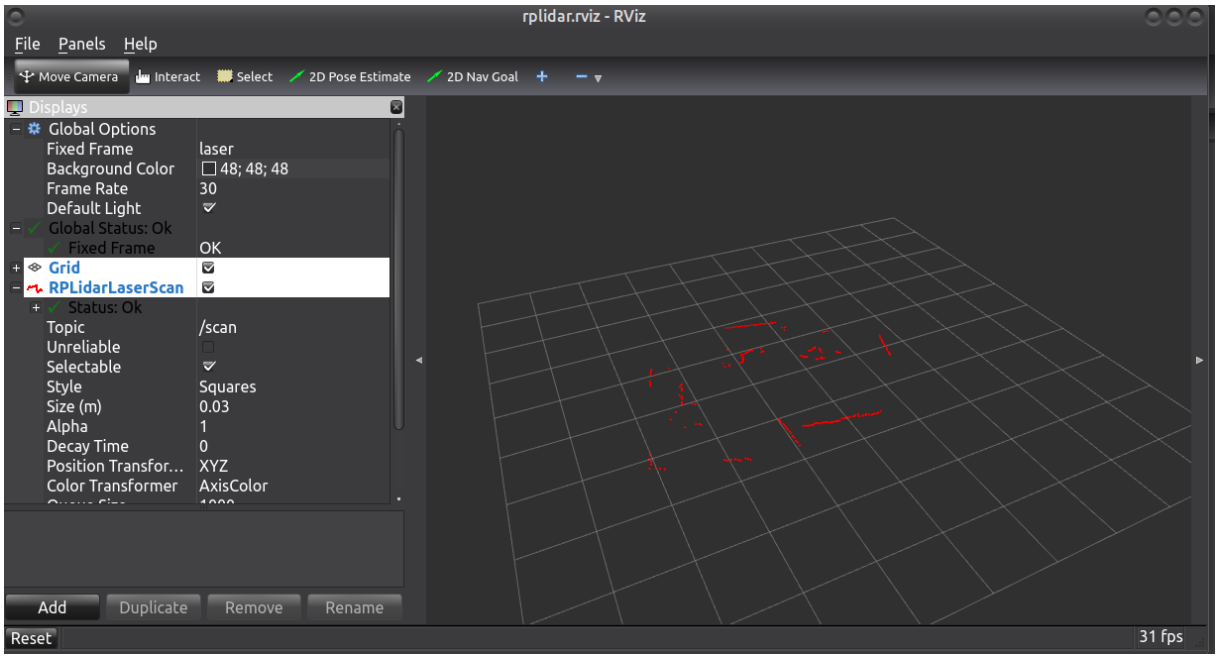
Figura 47. Leitura das mensagens recebidas



Fonte: (Autora do trabalho).

Quando não há obstáculos, a mensagem retorna *inf*, ou seja, infinito, indicando que não há um valor de distância correspondente ao *scan*. A partir do Rviz, é possível visualizar a leitura do escaneamento de forma gráfica. A imagem abaixo mostra a leitura correspondente às mensagens mostradas na Figura 47.

Figura 48. Mensagens do sensor no Rviz



Fonte: (Autora do trabalho).

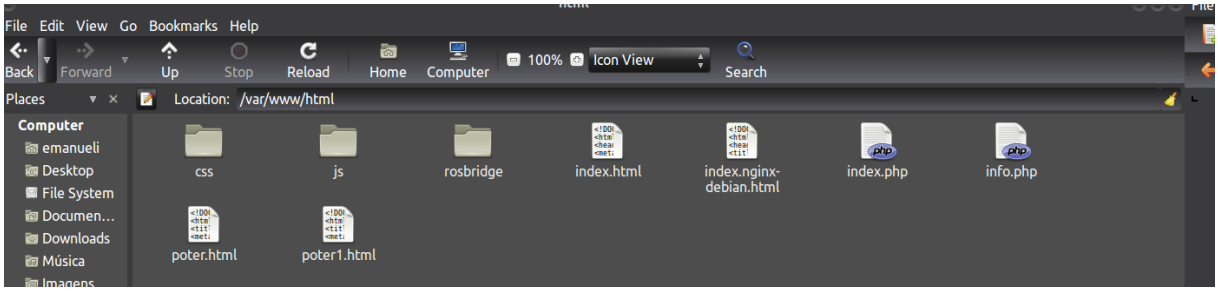
5.3.5 Interface de controle via WEB

A fim de possibilitar que usuários que não possuam ROS instalados em sua máquina possam também, utilizar o sistema desenvolvido para controle do robô, desenvolveu-se uma interface web, programada em HTML. Para tanto utilizou-se o *rosbridge*, ferramenta que oferece uma JSON API para abstrair a comunicação ROS com outros dispositivos. Fez-se a configuração do *Rosbridge_suite*, um pacote que contém o *rosbridge* e suporte para ferramentas de desenvolvimento *front end*. A instalação deste pacote foi realizada no Odroid, conforme (Rosbridge_suite).

Para o desenvolvimento da página, primeiramente fez-se a configuração de um servidor Apache, o qual é responsável por transformar o arquivo HTML estático em uma página navegável. Após a instalação do servidor, o arquivo de programação, bem

como os recursos necessários para sua exibição foram copiados na pasta *default* do servidor, a fim de que o mesmo possa realizar a leitura do arquivo e posterior exibição através do navegador do usuário (Figura 49).

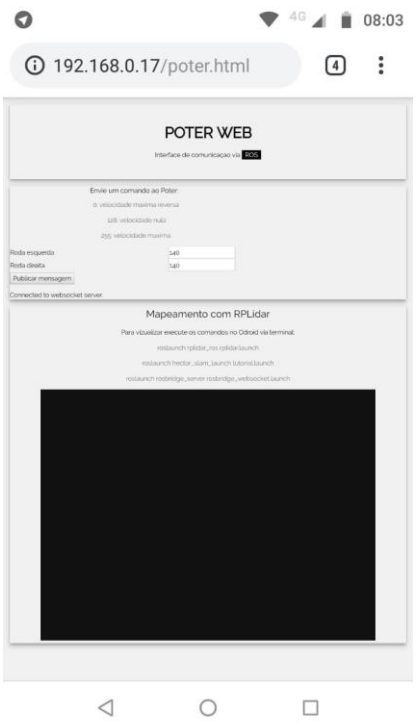
Figura 49. Pasta default do servidor Apache



Fonte: (Autora do trabalho).

Dessa forma, qualquer dispositivo conectado à rede tem acesso ao servidor e pode acessar o endereço da página diretamente por um navegador (Figura 50).

Figura 50. Acesso da página HTML pelo navegador de um dispositivo Android



Fonte: (Autora do trabalho).

A página foi programada para permitir o envio de comandos aos motores, monitoramento da detecção de distância pelos sensores ultrassônicos, bem como a visualização do mapeamento do LIDAR em tempo real. Para visualizar o mapeamento, fez-se ainda, a instalação do *ros3djs*.

Utilizou-se os recursos requeridos para processamento do código, adquiridos a partir da instalação da biblioteca *roslibjs.* Esta biblioteca é a base em JavaScript para realizar a interação entre o ROS e o navegador.

O código realizado para esta tarefa pode ser visto por completo no ANEXO D. Aqui serão discutidos os principais pontos do algoritmo. Primeiramente, fez-se a declaração das variáveis. A função new ROSLIB.Topic indica que a variável declarada será relacionada a um tópico ROS, sendo necessário declarar o nome relativo ao IP e à porta de direcionamento, o nome do tópico ao qual a variável deve corresponder, além do tipo de mensagem a ser utilizada na comunicação, apresentado na Figura 51.

Figura 51. Declaração de variáveis

```
var cmdVelEsquerda = new ROSLIB.Topic({
    ros : rbServer,
    name : '/cmd_left_wheel',
    messageType : 'std_msgs/Int16'
});

var cmdVelDireita = new ROSLIB.Topic({
    ros : rbServer,
    name : '/cmd_right_wheel',
    messageType : 'std_msgs/Int16'
});

var listener = new ROSLIB.Topic({
    ros : rbServer,
    name : '/distancia',
    messageType : 'std_msgs/String'
  });

  listener.subscribe(function(message) {
    console.log('Sensores ultrassonicos ' + listener.name + ': ' + message.data);
    listener.unsubscribe();
  });
```

Fonte: (Autora do trabalho).

Foram declarados os tópicos relacionados à velocidade dos motores, citados na seção 5.3.3 , bem como o tópico *distancia*, que será discutido na seção 5.4.2 . Ao tópico *distancia* foi declarado um *subscribe*, a fim de permitir ao navegador realizar a leitura destes dados.

Para realizar a leitura do mapeamento, foi utilizada a função ROS3D.Viewer, vinculado à identificação *map*, para que a página possa exibi-lo, conforme Figura 52.

Figura 52. Variáveis para leitura do mapeamento

```
function init() {

    // Create the main viewer.
    var viewer = new ROS3D.Viewer({
        divID : 'map',
        width : 800,
        height : 600,
        antialias : true
    });

// Setup the marker client.
    var gridClient = new ROS3D.OccupancyGridClient({
        ros : rbServer,
        rootObject : viewer.scene,
        continuous: true
    });
}
```

Fonte: (Autora do trabalho).

A Figura 53 apresenta a parte do algoritmo relativa à publicação das velocidades inseridas na página:

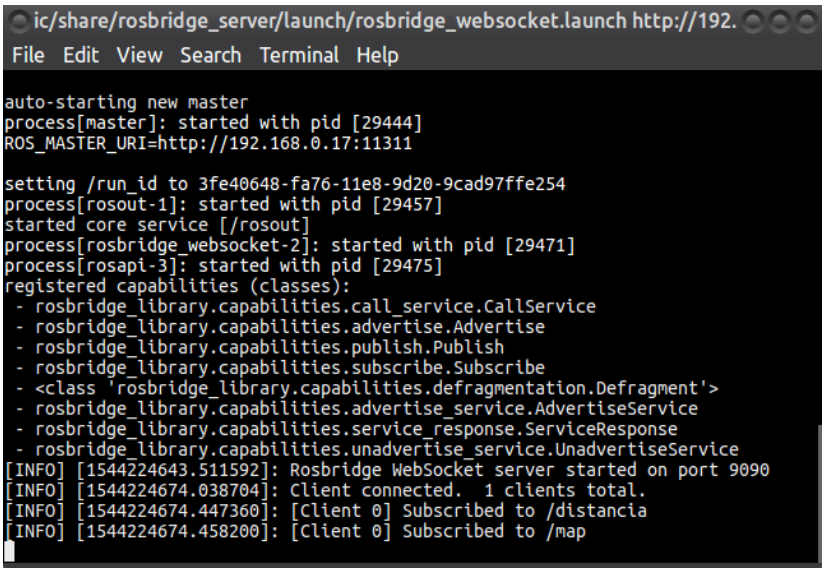
Figura 53. Publicação de valores de velocidade inseridos pelo usuário

```
function pubMessage() {
    /**
    Publica nos topicos relativos a velocidade dos motores de acordo com os valores inseridos
    pelo usuario na caixa de text
    **/

    var rodaEsquerda = 128;
    var rodaDireita = 128;

    // Leitura dos valores
    rodaEsquerda = Number(document.getElementById('RodaEsquerdaText').value);
    rodaDireita = Number(document.getElementById('RodaDireitaText').value);

    var cmdL = new ROSLIB.Message({
        data: rodaEsquerda
    });

    var cmdR = new ROSLIB.Message({
        data: rodaDireita
    });

    // Publica as mensagens
    cmdVelEsquerda.publish(cmdL);
    cmdVelDireita.publish(cmdR);
}
```

Fonte: (Autora do trabalho).

Ao iniciar o pacote *rosbridge* pelo terminal o console retorna o número de clientes conectados (páginas abertas no endereço do arquivo html), bem como a indicação de que esses podem subscrever os tópicos declarados (Figura 54).

Figura 54. Início da comunicação pelo pacote *rosbridge*



Fonte: (Autora do trabalho).

5.4 INTEGRAÇÃO HARDWARE E SOFTWARE

5.4.1 Controle dos motores via ROS

Os pinos de comunicação dos drivers dos motores são conectados diretamente ao Arduino, nas portas TX1 e RX1. Sendo assim, para realização de envio de comandos aos motores, utilizou-se o pacote *rosserial*, citado anteriormente em 5.3.3 , a fim de enviar comandos via serial para o Arduino para que esse então se comunique com os motores. O algoritmo básico de comando foi programado em C++ pela IDE do Arduino, inicialmente sendo realizado um teste de comando a partir do envio de dados via serial.

O Arduino deve ser capaz de subscrever os tópicos de velocidade dos motores, a fim de ler os dados enviados pela rede ROS, e enviá-los aos motores. Para tanto,

declarou-se as funções *cmdRightWheelCB* e *cmdLeftWheelCB*, as quais enviam as mensagens recebidas pelos tópicos *cmd_left_wheel* e *cmd_right_wheel* na comunicação serial com os drivers dos motores. As velocidades são setadas pelas funções *SET_SPEED1* e *SET_SPEED2* conforme datasheet do MD49. Parte do código é apresentado na Figura 55.

Figura 55. Algoritmo para comunicação com os motores

```
void cmdLeftWheelCB( const std_msgs::Int16& msg)
{
  Serial1.write(CMD);
  Serial1.write(SET_SPEED1);
  Serial1.write(msg.data);
}

void cmdRightWheelCB( const std_msgs::Int16& msg)
{
  Serial1.write(CMD);
  Serial1.write(SET_SPEED2);
  Serial1.write(msg.data);
}

ros::Subscriber<std_msgs::Int16> subCmdLeft("cmd_left_wheel", cmdLeftWheelCB );
ros::Subscriber<std_msgs::Int16> subCmdRight("cmd_right_wheel",cmdRightWheelCB );
```

Fonte: (Autora do trabalho).

Para realizar o envio de comandos, é necessário iniciar o mestre ROS, iniciar o nó de comunicação do Arduino no dispositivo Odroid, e, posteriormente, no computador mestre, é possível publicar e subscrever nos tópicos relacionados às velocidades dos motores, o que pode ser feito diretamente pelo terminal (Figura 56).

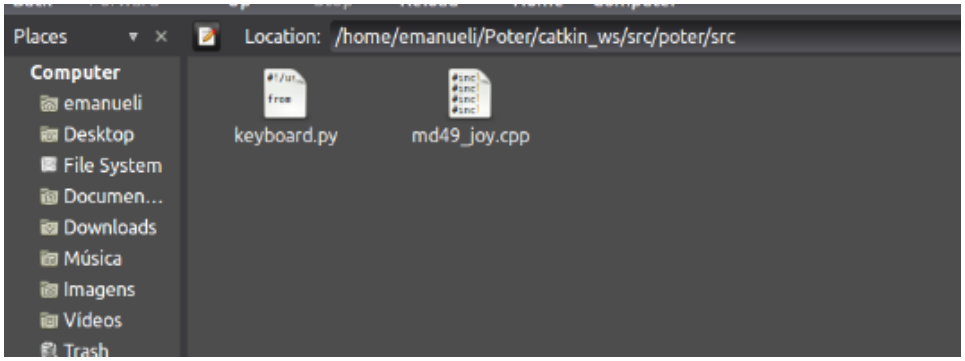
Figura 56. Envio de comandos manualmente pelo terminal

```
File  Edit  View  Search  Terminal  Help
emanueli@emanueli-X555UB:~$ rostopic pub /cmd_right_wheel std_msgs/Int16 "data:
128"
publishing and latching message. Press ctrl-C to terminate
```

Fonte: (Autora do trabalho).

Porém, dessa forma é necessário abrir vários terminais, visto que cada um permite publicar apenas um dado por vez a partir do comando *rostopic pub*. Assim, após testar-se a comunicação, fez-se um código em Python, de forma a permitir o controle do robô via teclado. Para tanto, foi necessário acrescentar um código fonte, vinculado a um novo nó, ao ambiente de trabalho Poter criado anteriormente (Figura 57).

Figura 57. Pasta *poter/src*



Fonte: (Autora do trabalho).

No código em python, que pode ser visto por completo no ANEXO C, primeiramente declarou-se a mensagem a ser enviada ao console (Figura 58), de forma a simplificar o entendimento da interface pelo usuário.

Figura 58. Mensagem do console

```
msg = """
Comandos Poter via teclado
---------------------------
Movimentos:
        w
   a    s    d

Parada:
        p

Alterar velocidade:
c : aumenta
v : reduz

CTRL-C para sair
"""

erro = """
Velocidade limite
---------------------------

"""
```

Fonte: (Autora do trabalho).

A partir das especificações dos motores comentadas na seção 0, tem-se que os motores param quando recebem um valor inteiro igual a 128, abaixo desse rotacionam no sentido horário (robô se move para frente) , e acima rotacionam no sentido anti-horário (robô se move para trás). Assim, declarou-se a função *moveBindings*, na qual a tecla “p” representa uma parada, tecla “w” velocidade de 140 (robô anda para frente) e tecla “s” velocidade de 120 (robô anda para trás). As teclas “d” e “a” foram declaradas para mudança de direção, em funções separadas. Para alteração de velocidade, declarou-se as teclas “c” e “v”, conforme pode ser visualizado no código apresentado na Figura 59.

Figura 59. Funções para mover o robô

```
35  moveBindings = {
36          'p':(128),
37          'w':(140),
38          's':(120),
39      }
40
41  speedBindings = {
42          'c':(4),
43          'v':(-4),
44      }
45
46  turnRight = {
47          'd':(118),
48      }
49
50  turnLeft = {
51          'a':(118),
52      }
53
54  def getKey():
55      tty.setraw(sys.stdin.fileno())
56      select.select([sys.stdin], [], [], 0)
57      key = sys.stdin.read(1)
58      termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
59      return key
```

Fonte: (Autora do trabalho).

A função *getKey* é responsável por identificar a tecla que está sendo pressionada pelo usuário e retorná-la em *key*. Após, tem-se a função principal do código, onde a partir da tecla pressionada se faz a publicação nos tópicos de velocidade, mostrado na Figura 60.

Figura 60. Publicação nos tópicos

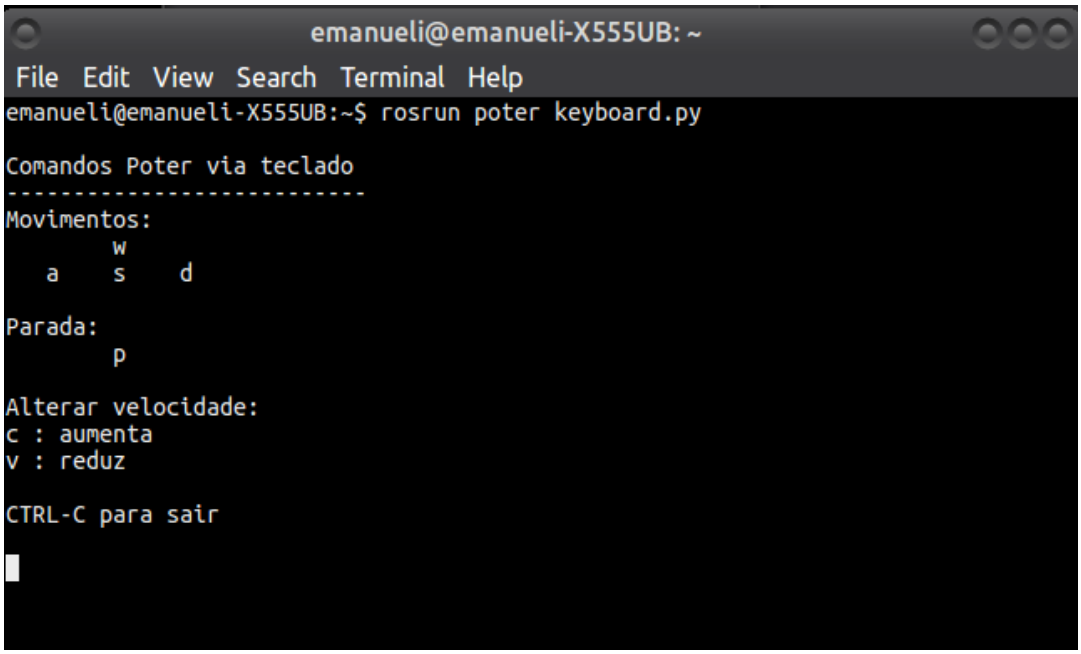
```
61  if __name__=="__main__":
62      settings = termios.tcgetattr(sys.stdin)
63
64      pub1 = rospy.Publisher('cmd_left_wheel', Int16, queue_size = 10)
65      pub = rospy.Publisher('cmd_right_wheel', Int16, queue_size = 10)
66      rospy.init_node('md49_keyboard')
67
68      right = 140;
69      left = 140;
70
71      try:
72          print(msg)
73          while(1):
74              key = getKey()
75
76              if key in moveBindings.keys():
77                  right = moveBindings[key]
78                  left = moveBindings[key]
79
80              elif key in speedBindings.keys():
81                  right = right + speedBindings[key]
82                  left = left + speedBindings[key]
83
84              elif key in turnRight.keys():
85                  right = turnRight[key]
87              elif key in turnLeft.keys():
88                  left = turnLeft[key]
89
90              else:
91                  key == '\x03'
92                  break
93
94              if right<152 and \
95                 right>110 and \
96                 left<152 and \
97                 left>110:
98                      pub.publish(right)
99                      pub1.publish(left)
00              else:
01                  print(erro)
02                  print(msg)
03
04
05      except Exception as e:
06          print(e)
07
08      finally:
09          termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
```

Fonte: (Autora do trabalho).

A função *rospy.Publisher* é vinculada separadamente aos tópicos de velocidade das rodas esquerda e direita. A publicação é realizada de acordo com a tecla pressionada, a partir de uma simples estrutura condicional. Caso a velocidade atinja uma velocidade acima ou abaixo dos limites suportadas pelo motor é exibido uma mensagem de erro. Caso seja pressionada uma tecla não declarada anteriormente, o processo é encerrado.

Para tornar o código executável, foi necessário executar o comando *chmod u+x keyboard.py*, a fim de liberar as permissões necessárias para o acesso ao mesmo. A Figura 64 mostra a interface em execução, após chamada do arquivo programado a partir do pacote *poter*.

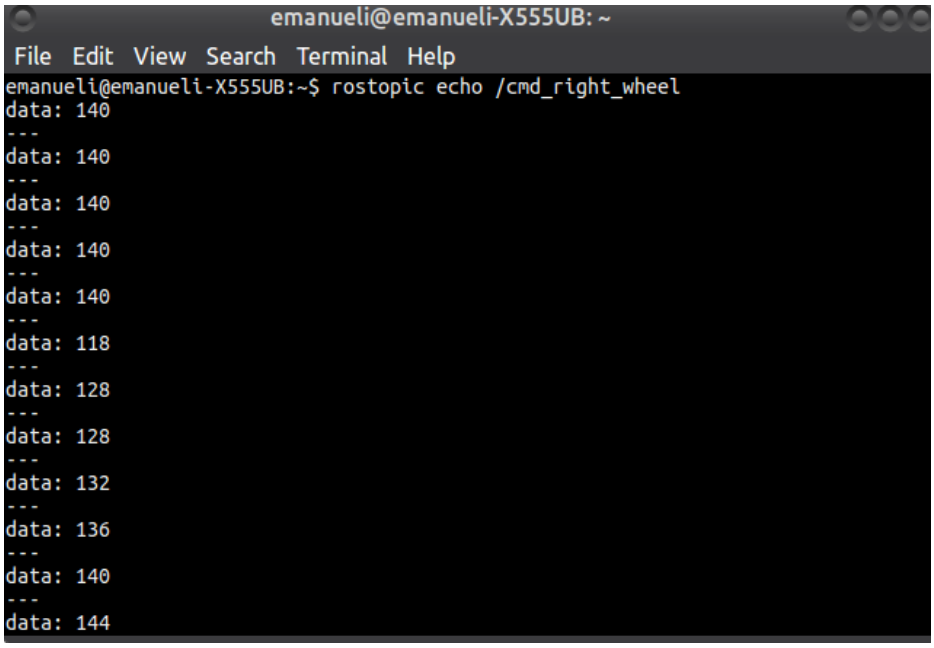
Figura 61. Interface de comando



Fonte: (Autora do trabalho).

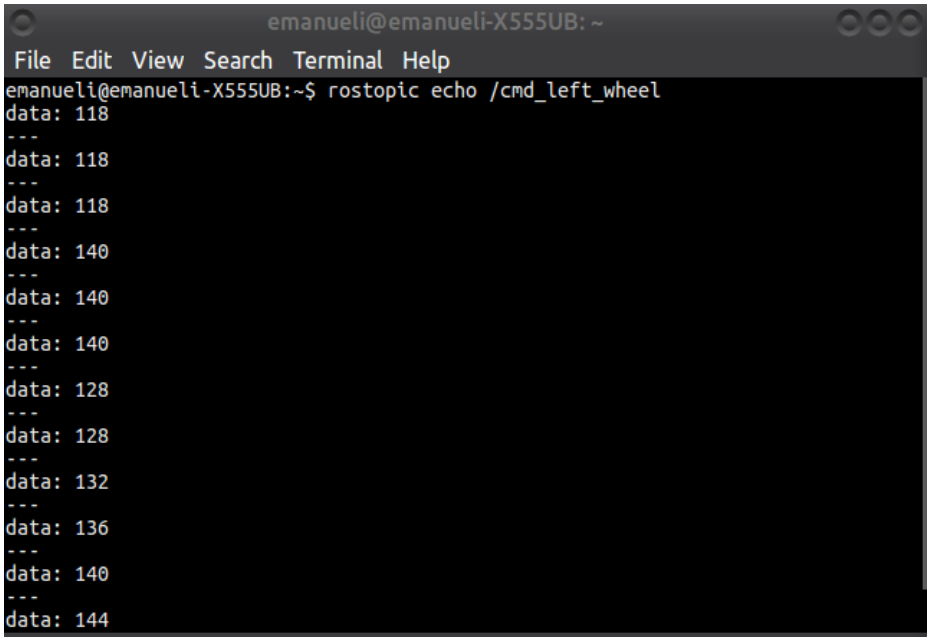
A publicação pode ser vista em outro terminal, conforme as Figuras 62 e 63.

Figura 62. Leitura do tópico de velocidade da roda direita



Fonte: (Autora do trabalho).

Figura 63. Leitura do tópico de velocidade da roda esquerda



Fonte: (Autora do trabalho).

5.4.2 Monitoramento dos sensores ultrassonicos via ROS

Como os sensores são conectados ao Arduino, a partir do shield projetado, também se utilizou do pacote *rosserial* para realizar a integração dos sensores ultrassônicos ao sistema. O código completo comentado pode ser visto no ANEXO B. Aqui serão discutidos os pontos principais do mesmo.

A fim de possibilitar a leitura dos dados em apenas um tópico, para simplificar o sistema, foi necessário declarar um *Publisher* para cada sensor, vinculando-os ao tópico *distancia* (Figura 64), o qual contém as distâncias retornada pelos sensores. Escolheu-se utilizar mensagens do tipo *string*, de forma a indicar de qual sensor é o dado obtido.

Figura 64. Declaração dos dados dos sensores ao tópico *distancia*

```
ros::NodeHandle  nh;
std_msgs::String sensor;
ros::Publisher chatter1("distancia",&sensor);
ros::Publisher chatter2("distancia",&sensor);
ros::Publisher chatter3("distancia",&sensor);
ros::Publisher chatter4("distancia",&sensor);
ros::Publisher chatter5("distancia",&sensor);
ros::Publisher chatter6("distancia",&sensor);
ros::Publisher chatter7("distancia",&sensor);
```

Fonte: (Autora do trabalho).

A distância lida pelo sensor é retornada em cm pela função *microsecondsToCentimeters*, e esta é transforma em *string*, declarando, por fim, a identificação do sensor em conjunto com o dado de leitura. Após, é dado um *delay* de 10µs, conforme mostrado na Figura 65.

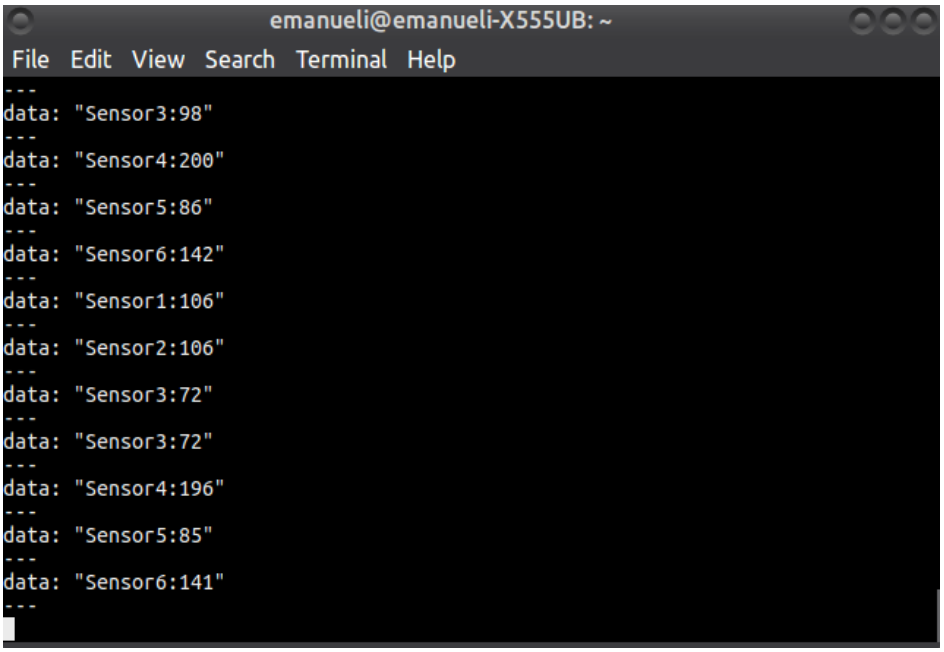
Figura 65. Conversão da distância em string

```
void loop()
{
  // O trigger recebe um pulso LOW por 2 microssegundos
  // e um HIGH por 5
  digitalWrite(pingTrigger1, LOW);
  delayMicroseconds(2);
  digitalWrite(pingTrigger1, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingTrigger1, LOW);
  // Verifica o tempo de duração do envio do sinal
  // deixando o pino echo em HIGH
  duration1 = pulseIn(Sensor1, HIGH);
  //Converte o tempo em distância
  cm1 = microsecondsToCentimeters(duration1);
  distancia1 = String(cm1);
  sensor1 =  String("Sensor1:" + distancia1);
  delay(10);
```

Fonte: (Autora do trabalho).

Tais declarações foram realizadas para todos os sensores. Na Figura 66 é possível visualizar o resultado obtido, via terminal:

Figura 66. Leitura dos dados dos sensores ultrassônicos

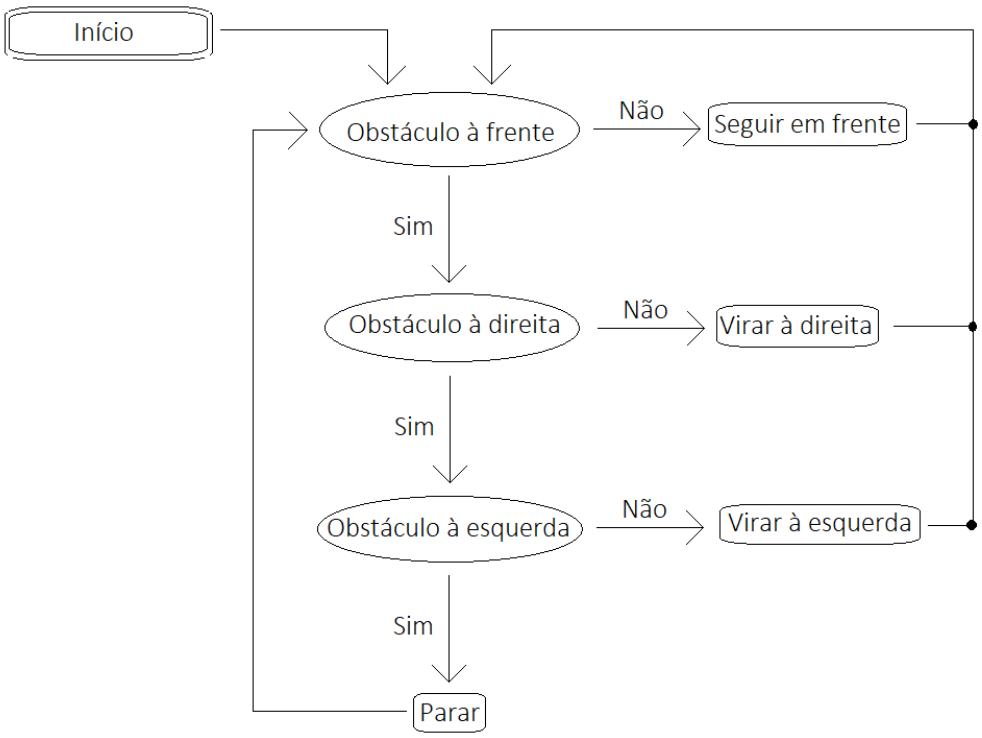


Fonte: (Autora do trabalho).

5.4.3 Algoritmo para detecção e desvio de obstáculos

Fez-se um simples algoritmo para realizar o desvio de obstáculos, a fim de que o robô navegasse de forma autônoma. O código pode ser visto no ANEXO B. Foram declaradas funções de movimento, para frente, parada, esquerda e direita, as quais são chamadas de acordo com os dados lidos pelos sensores (Figura 68). A partir de uma estrutura condicional (Figura 67), o robô para ou muda de direção quando percebe um obstáculo à frente.

Figura 67. Fluxograma de ação para desvio de obstáculos



Fonte: (Autora do trabalho).

Figura 68. Desvio de obstáculos

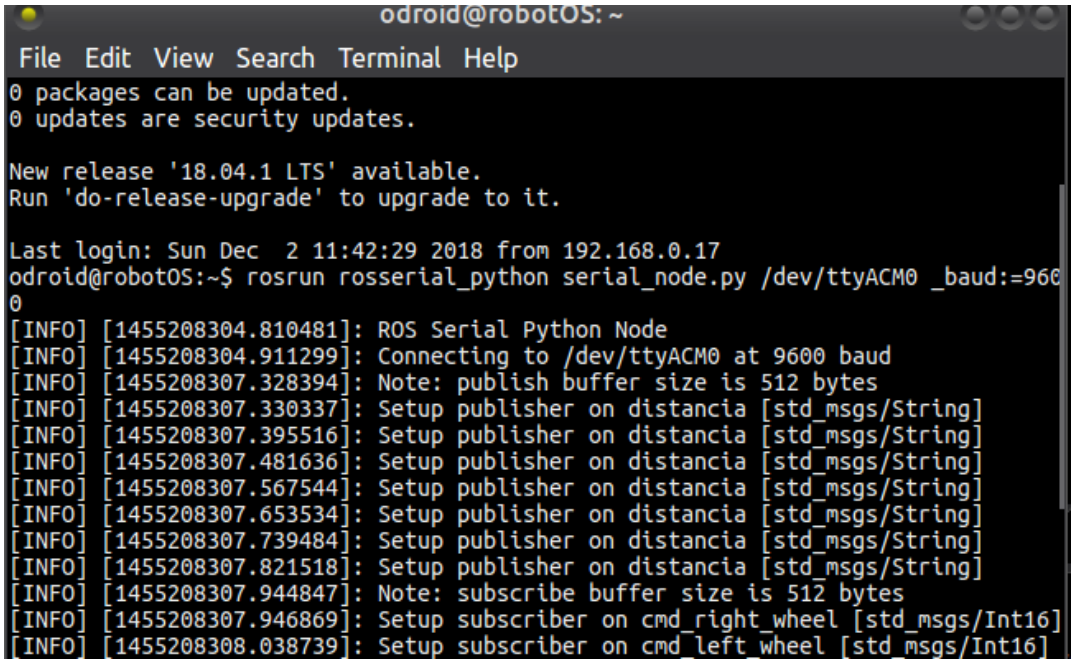
```
void frente()
{
  Serial1.write(CMD);
  Serial1.write(SET_SPEED1);
  Serial1.write(140);//mexer na velocidade da roda esquerda
  Serial1.write(CMD);
  Serial1.write(SET_SPEED2);
  Serial1.write(140);
}

void parar()
{
  Serial1.write(CMD);
  Serial1.write(SET_SPEED1);
  Serial1.write(128);//mexer na velocidade da roda esquerda
  Serial1.write(CMD);
  Serial1.write(SET_SPEED2);
  Serial1.write(128);//mexer na velocidade da roda  direita
}

 void virar_esquerda()
 {
   Serial1.write(CMD);
   Serial1.write(SET_SPEED1);
   Serial1.write(128);//mexer na velocidade da roda esquerda
   Serial1.write(CMD);
   Serial1.write(SET_SPEED2);
   Serial1.write(140);//mexer na velocidade da roda  direita
 }

 void virar_direita()
 {
   Serial1.write(CMD);
   Serial1.write(SET_SPEED1);
   Serial1.write(140);//mexer na velocidade da roda esquerda
   Serial1.write(CMD);
   Serial1.write(SET_SPEED2);
   Serial1.write(128);//mexer na velocidade da roda  direita
 }
```

```
if((cm2>=50)and(cm3>=50)and(cm4>=50)) //sem obstaculos
{
  frente();
}
if((cm1<20) and (cm7>10))
{
  virar_esquerda();
}
if((cm5<20) and (cm6>10))
{
  virar_direita();
}
if(((cm2<50)or(cm3<50)or(cm4<50)) and ((cm1<30)and(cm7>15)))
{
  virar_esquerda();
}else if (((cm2<50)or(cm3<50)or(cm4<50)) and ((cm5<30) and (cm6>15)))
{
  virar_direita();
}
if((cm2<10)or(cm3<10)or(cm4<10)or(cm1<10)or(cm2<10)or(cm6<10)or(cm7<10))
{
   parar();
}
```

Fonte: (Autora do trabalho).

Incluiu-se no código a condição de que caso o usuário enviasse um comando de parada via a interface citada anteriormente, o robô priorize tal comando para realizar os movimentos. Iniciando a comunicação pela interface serial, tem-se como retorno a possibilidade de leitura dos tópicos de distância e velocidade dos motores, conforme pode ser visualizado na Figura 69.

Figura 69. Comunicação via *rosserial*



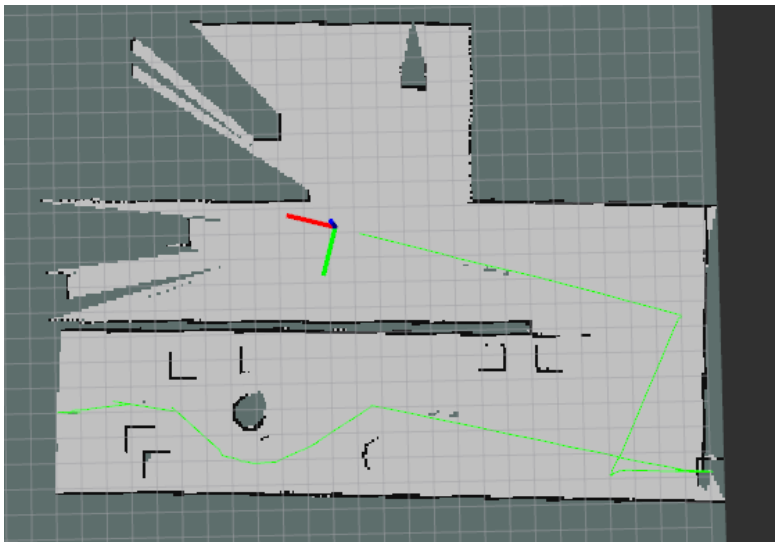
Fonte: (Autora do trabalho).

5.4.4 Mapeamento de ambientes com o RPLidar

Utilizando as mensagens do sensor de forma direta, nota-se que o mapeamento obtido pelo Rviz não permite uma visualização nítida do ambiente, conforme a Figura 48. Para contornar tal problema, as mensagens podem ser processadas pela ferramenta Hector Slam, a qual oferece um resultado aprimorado do mapeamento. Hector Slam é um pacote capaz de mapear o ambiente e, simultaneamente, estimar a posição 2D do sensor a laser. A configuração destes pacotes pode ser vista detalhadamente em (NICK).

A Figura 70 apresenta um exemplo de mapeamento realizado com o Hector Slam.

Figura 70. Hector Slam



Fonte: (ROS, 2014).

Os resultados obtidos para tal ferramenta a partir do robô Poter serão descritos no capítulo 6 .

6 RESULTADOS

A seguir serão discutidos os resultados finais obtidos a partir da integração do sistema desenvolvido.

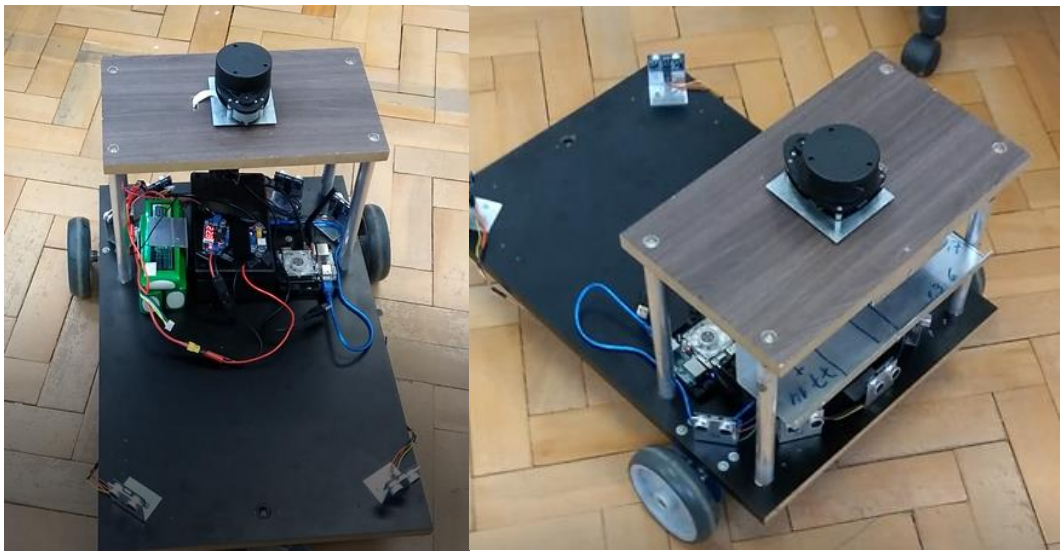
6.1 ADAPTAÇÃO DE HARDWARE

A confecção do *shield* foi um processo que exigiu precisão e cuidado. A transferência do projeto para placa de fenolite inicialmente não apresentou bom resultado, portanto o processo foi reiniciado até retornar uma resposta ótimo, na qual as trilhas estavam totalmente desenhadas, o que garantiu a proteção do cobre nessas áreas contra a corrosão no percloreto de ferro. O processo de soldagem foi constantemente testado com multímetro, dado a pequena distância entre os pinos é essencial evitar ligações equivocadas, a fim de prevenir a ocorrência de curto circuito. Devido ao tamanho dos pinos utilizados para conexão, inicialmente a placa apresentava mau contato com o Arduino, portanto, foi necessário aumentá-los. Após este ajuste nas conexões, a placa apresentou ótimo funcionamento, com boa resistência mecânica e ótima resposta aos comandos enviados.

O sistema de alimentação proposto apresentou bom funcionamento, atendendo às expectativas. As baterias utilizadas são de simples conexão e permitiram tornar o robô uma plataforma independente. O sistema foi testado com 4 horas de uso ininterrupto e as baterias não descarregaram, mesmo com o Odroid operando quase em carga máxima, com as 3 portas USB sendo utilizadas.

Ainda, as modificações no corpo do robô facilitaram a utilização da plataforma. Visto que o conjunto de sensores representa uma grande quantidade de cabos, a confecção do *shield* permitiu a organização destes, simplificando o manuseio do robô pelos membros do grupo GARRA. O resultado final pode ser visto na Figura 71.

Figura 71. Poter, visão traseira (à esquerda) e visão lateral (à direita)

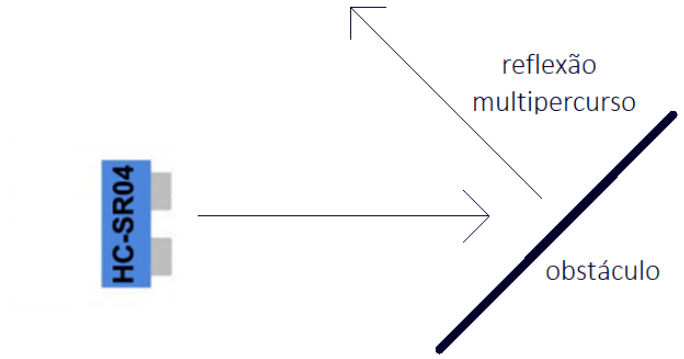


Fonte: (Autora do trabalho).

6.2 INTEGRAÇÃO HARDWARE-SOFTWARE

O sensor ultrassônico utilizado apresentou bons resultados nos testes apresentados na seção 5.2.1 , entretanto, quando os 7 operam em conjunto na plataforma, alguns às vezes retornam uma distância absurda (maior de 20m), diferindo à realidade. Esse comportamento se deve ao fato da onda mecânica se refletir em objetos que formam diferentes ângulos em relação ao transmissor e receptor. Isso implica em um retorno em multipercurso, o que indica um maior tempo de trajeto e justifica os valores altos retornados como resposta. Na Figura 72 é ilustrado este problema.

Figura 72. Obstáculo inclinado



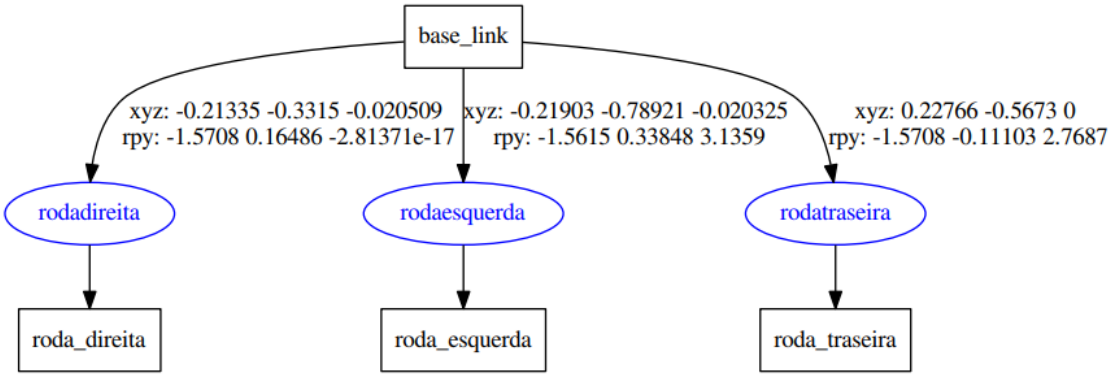
Fonte: (adaptado de FLOP).

Como as estruturas condicionais programadas para detecção e desvio de obstáculos não considera distância maiores que 50cm a navegação não é prejudica por esse comportamento. Entretanto, dependendo da distância e inclinação de um objeto, alguns sensores podem interferir na leitura de outro, inserindo erros nos dados obtidos. Para solucionar tais erros inseridos, em trabalhos futuros poderia se acrescentar um algoritmo de controle para esses dados, como, por exemplo, a tomada de decisão ser realizada após a média dos valores lidos por um mesmo sensor, a fim de confirmar a informação recebida.

O projeto do modelo 3D foi realizado para facilitar trabalhos futuros de simulação no ambiente ROS utilizando o robô Poter. A fim de validar o modelo 3D projetado, utilizou-se o comando *check_urdf* a partir do arquivo .urdf exportado pelo *plugin*.

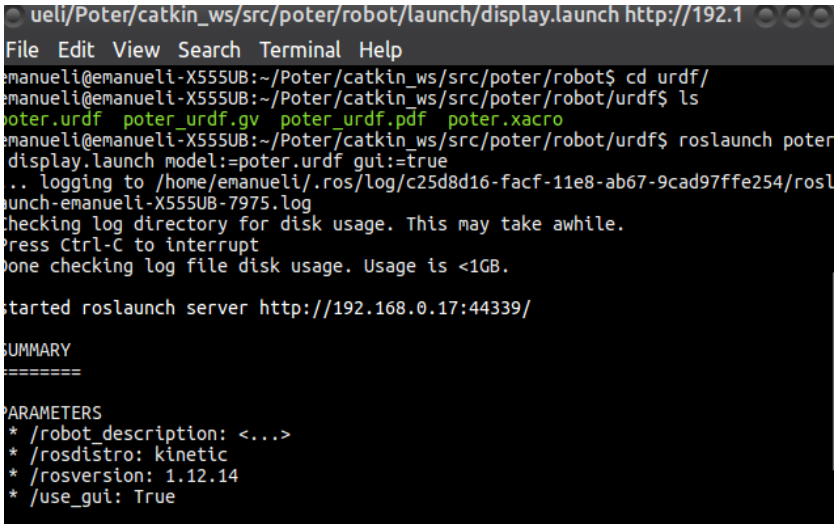
O corpo do robô foi chamado de base_link, sendo o principal do modelo. As rodas direita esquerda e traseira são *children* do *base_link*, estando vinculadas ao mesmo (Figura 73). Para verificar o posicionamento e grau de liberdade dos movimentos do robô, fez-se a simulação do modelo do Poter no Rviz. O comando realizado foi conforme a Figura 74.

Figura 73. Modelo 3D – links



Fonte: (Autora do trabalho).

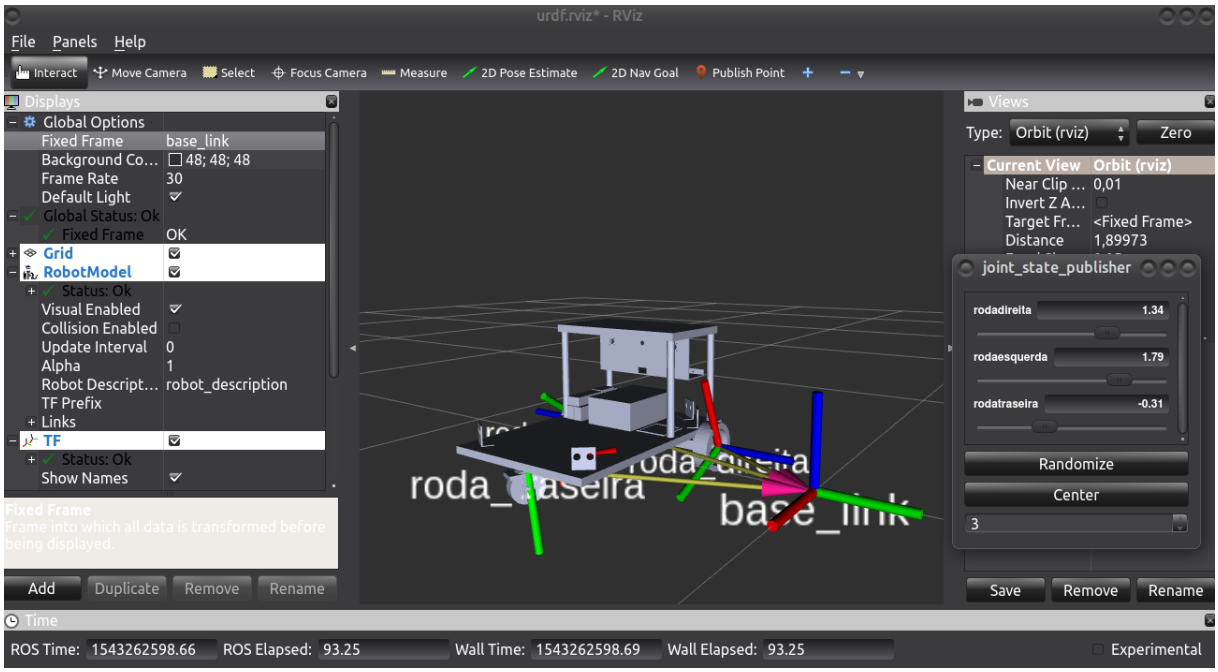
Figura 74. Comando para visualização do modelo no Rviz



Fonte: (Autora do trabalho).

A resposta obtida pode ser visualizada Figura 75. A partir do *joint_state_publisher*, foi possível verificar o movimento de cada roda. Os resultados foram satisfatórios e corresponderam ao modelo projetado. Entretanto, o Rviz não foi capaz de captar as cores do modelo exportadas pelo plugin, as chamadas *meshes*. Dessa forma, para uma fiel representação seria necessário programar as *meshes* manualmente ou utilizar outro *plugin* para gerá-las automaticamente.

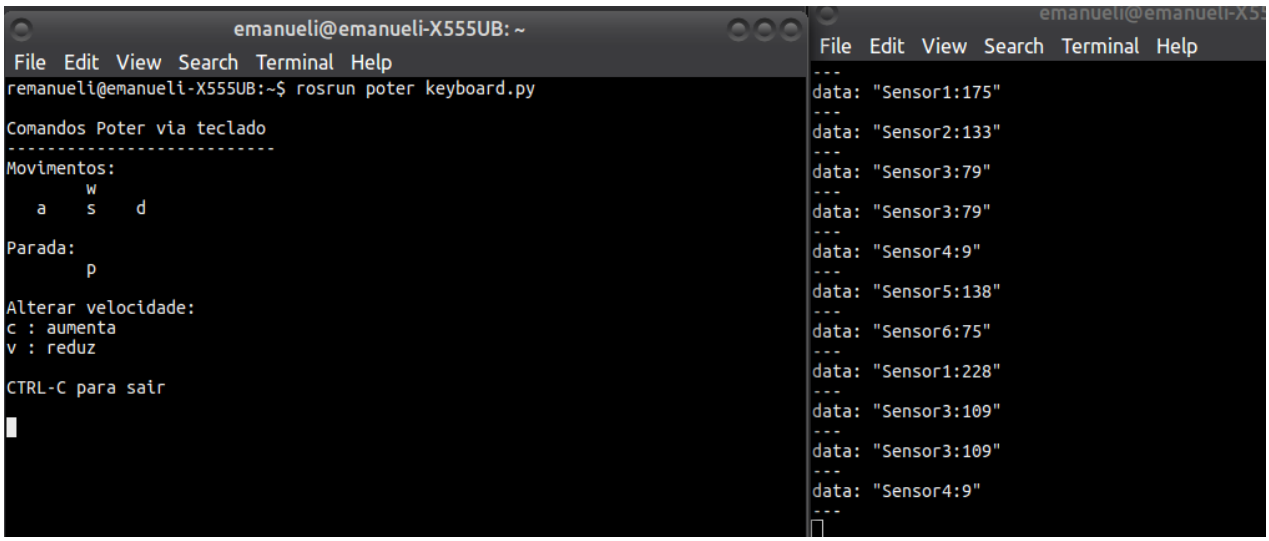
Figura 75. Modelo do Poter no Rviz



Fonte: (Autora do trabalho).

A interface de comando para controle do robô pelo teclado (Figura 76) apresentou bom funcionamento, até mesmo para realizar alteração de velocidade nos motores. Em conjunto com o código de navegação autônoma, conforme ANEXO B, quando a tecla “p” é pressionada o robô para.

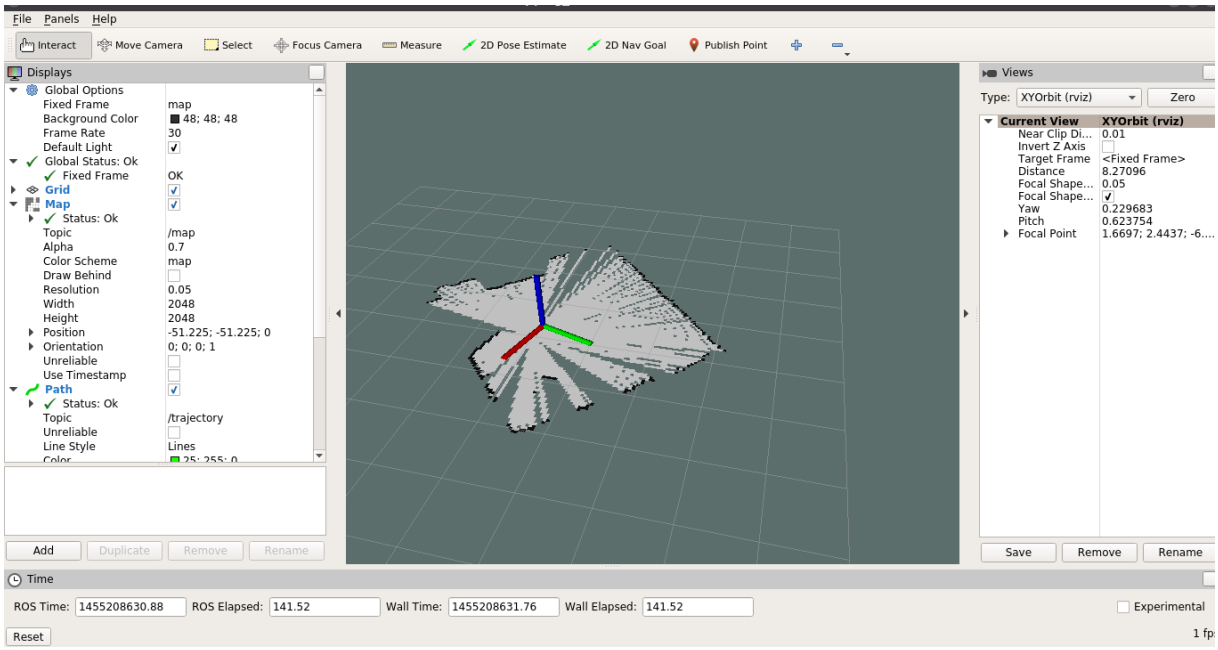
Figura 76. Interface de comando e leitura dos sensores



Fonte: (Autora do trabalho).

O teste realizado com o *hector_slam* apresentou bom resultado, justificando seu uso pelo comunidade. A partir dos dados adquiridos pelo pacote *rp_lidar* o pacote *hector* constrói o mapa do ambiente, bem como mostra a localização e trajetória do sensor realizada durante o mapeamento (Figura 77).

Figura 77. Teste de funcionamento do *hector_slam*

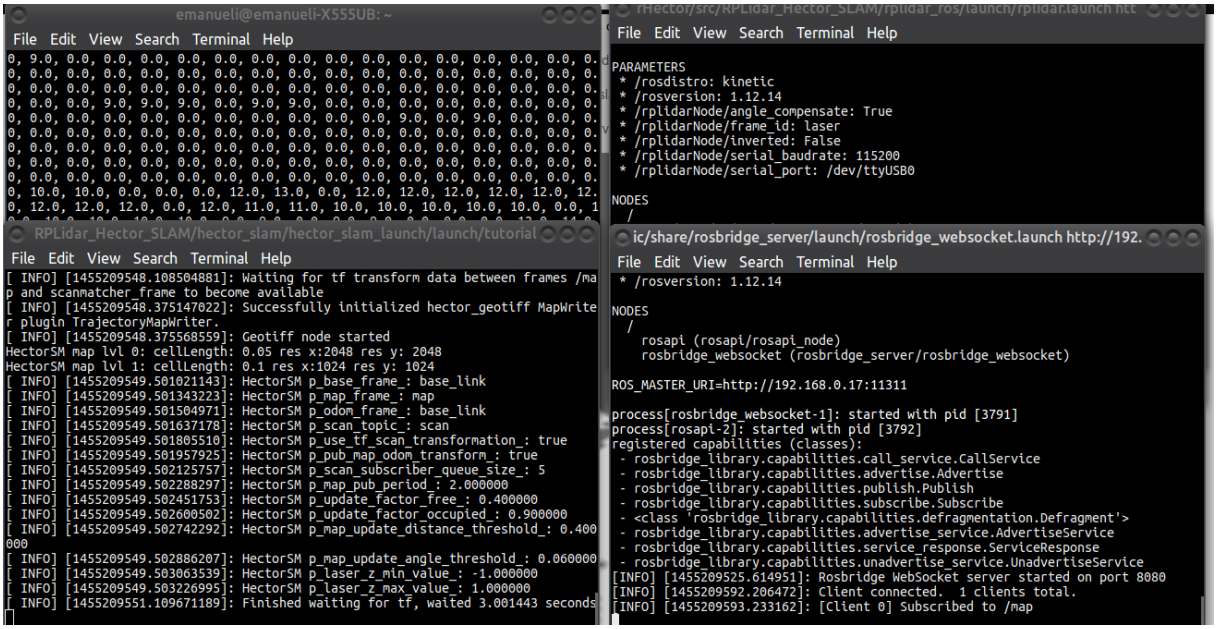


Fonte: (Autora do trabalho).

A página em *html* permitiu uma alternativa para integração com o navegador. O envio de comandos é realizado por meio de um botão na página, quando esse é pressionado. A página permite a visualização dos dados dos sensores no console do navegador e o mapeamento do LIDAR quando o pacote *hector_slam* é executado. Entretanto, para o funcionamento da página é necessário sempre iniciar o pacote *rosbridge* no Odroid, bem como os pacotes *rp_lidar* e *hector_slam* para visualização do mapa. Além disso, a visualização do mapeamento em tempo real é prejudicada, devido à baixa resolução e velocidade de atualização da biblioteca utilizada. Observou-se que a visualização do mapeamento apresenta melhores resultados diretamente no Rviz.

A Figura 78 mostra os comandos necessários para funcionamento completo da página:

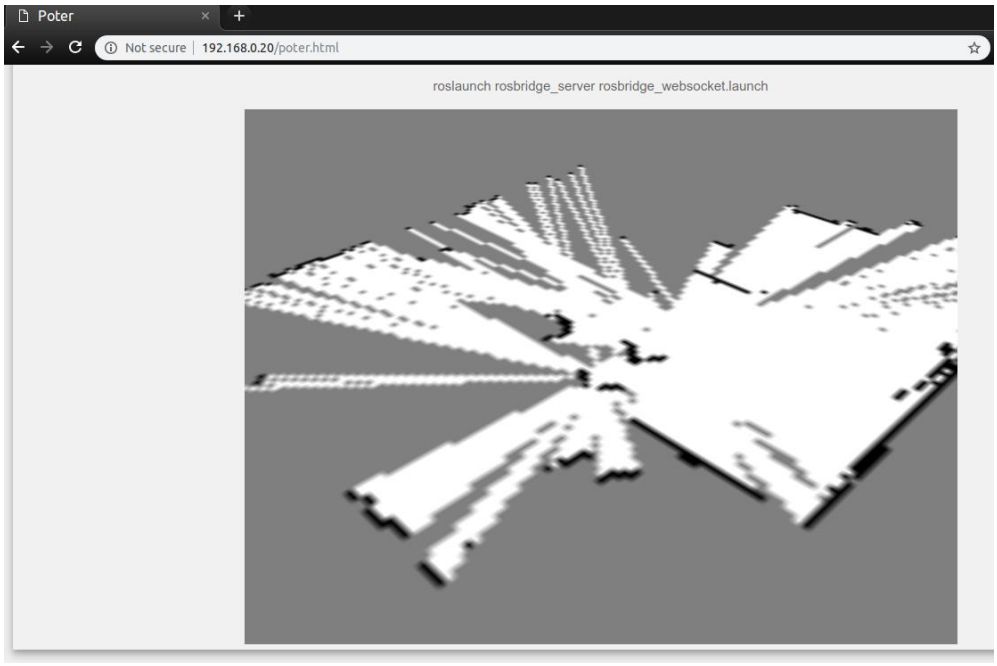
Figura 78. Comandos para funcionamento completo da página



Fonte: (Autora do trabalho).

Resultado do mapeamento obtido pela página web é apresentado na Figura 79.

Figura 79. Mapeamento visto pela página HTML



Fonte: (Autora do trabalho).

7 CONCLUSÃO

O projeto desenvolvido apresentou o desenvolvimento de um sistema embarcado com o Odroid-XU4 para realizar a integração sensorial de um veículo terrestre não tripulado. As alterações físicas realizadas na plataforma robótica foram de extrema importância para a construção do sistema. A integração dos componentes eletrônicos, principalmente a partir da confecção do *shield* e do sistema de alimentação a partir das baterias tornou o robô independente, facilitando os trabalhos desenvolvidos no mesmo.

O dispositivo Odroid apresentou bons resultados para o fim à que foi destinado, visto que a velocidade de processamento desse foi satisfatória mesmo após a interação dos sensores e do sistema de comunicação remota. Se destaca a importância de testes, medições e monitoramento do funcionamento da plataforma a fim de validar os componentes do sistema, bem como realizar as atualizações necessárias ao embarcado.

Após a implementação do sistema, notou-se a facilidade do desenvolvimento de ferramentas na plataforma ROS, bem como o grande acervo disponível de pacotes para diversos sensores e componentes. Para trabalhos futuros pode-se explorar os dados obtidos pelo sensor LIDAR, aprimorando o algoritmo de desvio de obstáculos. Ademais, pode-se utilizar o modelo 3D projetado, de forma a integrar as ferramentas de simulação disponibilizadas pelo ROS.

Visto que foi realizada a implementação do Odroid na plataforma e fez-se a integração dos componentes de hardware e software, e ambos apresentaram um bom funcionamento, pode-se afirmar que os principais objetivos do trabalho foram atingidos, corroborando a metodologia utilizada.

8 REFERÊNCIAS

ALMEIDA. Microcontrollers. **Roboticaclab**, 2014. Disponivel em: <http://home.roboticlab.eu/pt/microcontrollers>. Acesso em: 30 mar. 2018.

ALVES, I. **Biblioteca Digital**. Disponivel em: <http://www.bibliotecadigital.funvicpinda.org.br:8080/jspui/bitstream/123456789/275/1/igorALVES.pdf>. Acesso em: 30 mar. 2018.

ARDUINO. Arduino MEGA 2560. **Arduino Store**, 2018. Disponivel em: <https://store.arduino.cc/usa/arduino-mega-2560-rev3>.

AURELIANO, A. Microcontroladores. **Fiozera**, 20117. Disponivel em: <https://fiozera.com.br/microcontroladores-914a59cbf7de>. Acesso em: 30 mar. 2018.

BEAL, V. Single Board Computer. **Webopedia**. Disponivel em: <www.webopedia.com/TERM/S/sbc_single_board_computer.html>. Acesso em: 30 mar. 2018.

BORBA, G. Página pessoal. **UFTPR**, 2017. Disponivel em: <http://paginapessoal.utfpr.edu.br/gustavobborba/material/files/mc_nocoesGerais.pdf>. Acesso em: 30 mar. 2018.

DENIS FERNANDO WOLF, E. D. V. S. Robótica Móvel Inteligente. **Osorio**, 2009. Disponivel em: <http://osorio.wait4.org/publications/2009/CL_JAI2009_Completo.pdf>. Acesso em: 17 Setembro 2018.

ELECTRONICS, R. EMG49. **Robot Electronics**. Disponivel em: <https://www.robot-electronics.co.uk/htm/emg49.htm>. Acesso em: 02 maio 2018.

ELECTRONICS, R. Robot Electronics. **MD49 - Dual 24 Volt 5 Amp H Bridge Motor Drive**. Disponivel em: <http://www.robot-electronics.co.uk/htm/md49tech.htm>. Acesso em: 02 maio 2018.

ELETRONICS, R. EMG49, mounting bracket and wheel specification. **EMG49**. Disponivel em: <http://www.robot-eletronics.co.uk/htm/emg49.htm>. Acesso em: 13 Agosto 2018.

FLOP, F. Sensor de Distância Ultrassônico HC-SR04. **Filipe Flop**. Disponivel em: <https://www.filipeflop.com/produto/sensor-de-distancia-ultrassonico-hc-sr04/#tab-accessories>.

GASPARI, S. Coenc 2014. **Xbot**, 2014. Disponivel em: <http://www.xbot.com.br/wp-content/uploads/2012/10/PB_COENC_2014_2_02.pdf>. Acesso em: 02 maio 2018.

GUSTAVO TEIXEIRA, L. T. M. G. R. D. V. R. F. E. V. R. Robótica em um contexto social. **Research Gate**. Disponivel em: <https://www.researchgate.net/publication/285598759_Robotica_em_um_contexto_social>. Acesso em: 19 set. 2018.

HARDKERNEL. Products. **Odroid U3**, 2015. Disponivel em: <https://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825&tab_idx=1>. Acesso em: 2 Jun 2018.

HOSTINGER. Como funciona o SSH. **Hostinger**. Disponivel em: <https://www.hostinger.com.br/tutoriais/como-funciona-o-ssh/>. Acesso em: 9 jun. 2017.

ITEADSTUDIO. Ultrasonic ranging module. **IteadStudio**, 2010. Disponivel em: <ftp://imall.iteadstudio.com/Modules/IM120628012_HC_SR04/DS_IM120628012_HC_SR04.pdf>. Acesso em: 20 jun. 2018.

JOSEPH, L. **Mastering ROS for Robotics Programming**. [S.l.]: Packt Publishing, 2015.

JOSEPH, L. **Ros Robotics Projects**. [S.l.]: Packt Publishing, 2017.

JUNIOR, F. Entendendo as mensagens e tópicos do ROS. **Embarcados**, 2016. Disponivel em: <https://www.embarcados.com.br/entendendo-as-mensagens-e-topicos-do-ros/>. Acesso em: 10 mar. 2018.

NASCIMENTO, E. J. D. RECONHECIMENTO DE GESTOS EM IMAGENS UTILIZANDO UM. [S.l.]: [s.n.], 2017.

NICK. GitHub. **RPLidar_Hecto_Slam**. Disponivel em: <https://github.com/NickL77/RPLidar_Hector_SLAM>. Acesso em: 5 jul. 2018.

PIO, J. L. S.; CASTRO, T. H.; C. A robótica móvel como instrumento de apoio à aprendizamgem de computação. **XVII - Simpósio brasileiro de informática na educação**, Brasília, 2006.

PYO, Y. **ROS Robot Programming**. Seoul, Repuclib of Kores: ROBOTIS Co, 2017.
R., R. Odroid XU4 User Manual. **Odroid Magazine**. Disponivel em: <https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf>. Acesso em: 05 Junho 2018.

ROBOPEAK. Robopeak. **GitHub**. Disponivel em: <https://github.com/robopeak/rplidar_ros/wiki>. Acesso em: 5 set. 2018.

ROS, W. Tutorials. **Wiki ROS**, 2014. Disponivel em: <http://wiki.ros.org/ROS/Tutorials>. Acesso em: 10 mar. 2018.

ROS, W. Kinetic. **Wiki ROS**. Disponivel em: <http://wiki.ros.org/kinetic/Installation/Ubuntu>. Acesso em: 5 jan. 2018.

ROSBRIDGE_SUITE. **Wiki ROS**. Disponivel em: <http://wiki.ros.org/rosbridge_suite>. Acesso em: 30 nov. 2018.

ROY, R. Getting start with the Odroid u3. **Odroid Magazine**, South Korea, p. 4-6, 2014.
SAMPAIO, C. Embarcados. **Odroid U3**, 2015. Disponivel em: <https://www.embarcados.com.br/odroid-u3/>. Acesso em: 9 Jun 2017.

SANTOS, P. H. M. **Desenvolvimento de um robô móvel autônomo de alta velocidade**. USP. São Carlos. 2015.

SECCHI, H. **Uma Introdução aos Robôs Móveis**. Argentina: INAUT, 2008.

SINGLE-BOARD Computer. **Techopedia**. Disponivel em: <https://www.techopedia.com/definition/9266/single-board-computer-sbc>. Acesso em: 30 mar. 2018.

SLAMTEC. RPLIDAR A1 Low Cost 360 Degree Laser Range Scanner Introduction and Datasheet. **SLAMTEC**, 2016. Disponivel em: <https://www.robotshop.com/media/files/pdf/rplidar-a1m8-360-degree-laser-scanner-development-kit-datasheet-1.pdf>. Acesso em: 18 Setembro 2018.

SOUZA, F. Embarcados. **Arduino MEGA 2560**, 2014. Disponivel em: <https://www.embarcados.com.br/arduino-mega-2560/>. Acesso em: 2 Outubro 2017.

WIKI, I. Ultrasonic Ranging Module HC-SR04. **Itead Wiki**. Disponivel em: <https://www.itead.cc/wiki/Ultrasonic_Ranging_Module_HC-SR04>. Acesso em: 19 Agosto 2018.

WIKIPEDIA. LIDAR. **Wikipedia**. Disponivel em: <https://pt.wikipedia.org/wiki/LIDAR>. Acesso em: 18 Setembro 2018.

ZAMAIA, J. F. P. Sistema de Navegação Autônoma para Plataforma Robótica Móvel com Restrições Não-Holonômicas.

PIO, J. L. S.; CASTRO, T. H. C A Robótica Móvel como Instrumento de Apoio à Aprendizagem de Computação. Anais: XVII – Simpósio Brasileiro de Informática na Educação. UNB. Brasília: 2006

ANEXO A – Código sensor HCSR-04

```
// Definição dos pinos de conexão echo e trigger do sensor
const int pingTrigger = 8;
const int Sensor_1 = 9;
// Variáveis para cálculo da distância do osbtáculo
long cm1;
long duration1;
long microsecondsToCentimeters(long microseconds)
{
  // A velocidade do som é 340 m/s ou 29 microssegundos por centímetro
  // Deve se considerar que o sinal via e volta
  // portanto deve se considerar apenas metade do tempo de viagem do sinal
  return microseconds / 29 / 2;
}
void setup() {
   pinMode(pingTrigger, OUTPUT);
   pinMode(Sensor_1, INPUT);
  // Inicia a comucação serial
  Serial.begin(9600);
}
void loop()
{
  // Variáveis de duração do ping
  // 2 microssegundos em nível baixo
  // 5 microssegundos em nível alto
  digitalWrite(pingTrigger, LOW);
  delayMicroseconds(2);
  digitalWrite(pingTrigger, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingTrigger, LOW);
  // Para estimar a distância se considera o tempo
  // do  pino ECHO em nível alto
  duration1 = pulseIn(Sensor_1, HIGH);
  // Converte o tempo em distância
  cm1 = microsecondsToCentimeters(duration1);
  delay(100);
    Serial.print("Sensor 1:");
    Serial.print(cm1);
    Serial.print("cm\n");
}
```

ANEXO B – Código para detecção e desvio de obstáculos

```
#include <ros.h>                    //biblioteca ros
#include <std_msgs/Int16.h>
#include <std_msgs/Float64.h>
#include <std_msgs/String.h>
#include <SoftwareSerial.h>          //biblioteca para o md49
//definição dos endereços para comunicação com o driver
#define CMD       (byte)0x00         //endereço para envio de comandos ao motor
#define GET_VER         0x29
#define GET_ENC1        0x23
#define GET_ENC2        0x24
#define GET_VI          0x2C
#define GET_ERROR       0x2D
#define SET_ACCEL       0x33
#define SET_SPEED1      0x31
#define SET_SPEED2      0x32
#define RESET_ENCODERS  0x35
#define DISABLE_TIMEOUT  0X38            //desabilita timeout (parar quando não há comunicação por mais de 2 segundos)

//definição dos pinos dos sensores
const int pingTrigger1 = 24;
const int Sensor1 = 26;
const int pingTrigger2 = 28;
const int Sensor2 = 30;
const int pingTrigger3 = 46;
const int Sensor3 = 48;
const int pingTrigger4 = 34;
const int Sensor4 = 36;
const int pingTrigger5 = 38;
const int Sensor5 = 40;
const int pingTrigger6 = 42;
const int Sensor6 = 44;
const int pingTrigger7 = 50;
const int Sensor7 = 52;

long cm1;
String sensor1;
String distancia1;
char s1[22];
long duration1;
long cm2;
String sensor2;
String distancia2;
char s2[22];
long duration2;
long cm3;
String sensor3;
String distancia3;
```

```
char s3[22];
long duration3;
long cm4;
String sensor4;
String distancia4;
char s4[22];
long duration4;
long cm5;
String sensor5;
String distancia5;
char s5[22];
long duration5;
long cm6;
String sensor6;
String distancia6;
char s6[22];
long duration6;
long cm7;
String sensor7;
String distancia7;
char s7[22];
long duration7;

int comando_py=140;
//tempo de leitura dos sensores
long microsecondsToCentimeters(long microseconds)
{
  return microseconds / 29 / 2;
}
//declaração ddos plublishers para um mesmo nó
ros::NodeHandle  nh;
std_msgs::String sensor;
ros::Publisher chatter1("distancia",&sensor);
ros::Publisher chatter2("distancia",&sensor);
ros::Publisher chatter3("distancia",&sensor);
ros::Publisher chatter4("distancia",&sensor);
ros::Publisher chatter5("distancia",&sensor);
ros::Publisher chatter6("distancia",&sensor);
ros::Publisher chatter7("distancia",&sensor);

//funções para envio de comando aos motores via ROS
//retorna um valor inteiro
void cmdLeftWheelCB( const std_msgs::Int16& msg)
{
  Serial1.write(CMD);
  Serial1.write(SET_SPEED1);
  Serial1.write(msg.data);
  return comando_py;
}

void cmdRightWheelCB( const std_msgs::Int16& msg)
```

```
{
  Serial1.write(CMD);
  Serial1.write(SET_SPEED2);
  Serial1.write(msg.data);
  return comando_py;
}

//funções para navegação autonoma
void frente()
{
  Serial1.write(CMD);
  Serial1.write(SET_SPEED1);
  Serial1.write(140);//mexer na velocidade da roda esquerda
  Serial1.write(CMD);
  Serial1.write(SET_SPEED2);
  Serial1.write(140);
}

void parar()
{
  Serial1.write(CMD);
  Serial1.write(SET_SPEED1);
  Serial1.write(128);//mexer na velocidade da roda esquerda
  Serial1.write(CMD);
  Serial1.write(SET_SPEED2);
  Serial1.write(128);//mexer na velocidade da roda  direita
}

void virar_esquerda()
{
  Serial1.write(CMD);
  Serial1.write(SET_SPEED1);
  Serial1.write(128);//mexer na velocidade da roda esquerda
  Serial1.write(CMD);
  Serial1.write(SET_SPEED2);
  Serial1.write(140);//mexer na velocidade da roda  direita
}

void virar_direita()
{
  Serial1.write(CMD);
  Serial1.write(SET_SPEED1);
  Serial1.write(140);//mexer na velocidade da roda esquerda
  Serial1.write(CMD);
  Serial1.write(SET_SPEED2);
  Serial1.write(128);//mexer na velocidade da roda  direita
}
//habilita a leitura da velocidade enviada por ROS e a envia às
// cmdLeftWheelCB e cmdRightWheelCB
ros::Subscriber<std_msgs::Int16> subCmdLeft("cmd_left_wheel", cmdLeftWheelCB );
ros::Subscriber<std_msgs::Int16> subCmdRight("cmd_right_wheel",cmdRightWheelCB );
```

```
void setup() {
  Serial.begin(9600);
  Serial1.begin(9600);
  Serial1.write(CMD);
  Serial1.write(DISABLE_TIMEOUT);
  pinMode(pingTrigger1, OUTPUT);
  pinMode(Sensor1, INPUT);
  pinMode(pingTrigger2, OUTPUT);
  pinMode(Sensor2, INPUT);
  pinMode(pingTrigger3, OUTPUT);
  pinMode(Sensor3, INPUT);
  pinMode(pingTrigger4, OUTPUT);
  pinMode(Sensor4, INPUT);
  pinMode(pingTrigger5, OUTPUT);
  pinMode(Sensor5, INPUT);
  pinMode(pingTrigger6, OUTPUT);
  pinMode(Sensor6, INPUT);
  pinMode(pingTrigger7, OUTPUT);
  pinMode(Sensor7, INPUT);
  Serial.begin(9600);
  nh.initNode();
  nh.subscribe(subCmdRight);
  nh.subscribe(subCmdLeft);
  nh.advertise(chatter1);
  nh.advertise(chatter2);
  nh.advertise(chatter3);
  nh.advertise(chatter4);
  nh.advertise(chatter5);
  nh.advertise(chatter6);
  nh.advertise(chatter7);
}

void loop()
{
  // O trigger recebe um pulso LOW por 2 microssegundos
  // e um HIGH por 5
  digitalWrite(pingTrigger1, LOW);
  delayMicroseconds(2);
  digitalWrite(pingTrigger1, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingTrigger1, LOW);
  // Verifica o tempo de duração do envio do sinal
  // deixando o pino echo em HIGH
  duration1 = pulseIn(Sensor1, HIGH);
  //Converte o tempo em distância
  cm1 = microsecondsToCentimeters(duration1);
  distancia1 = String(cm1);
  sensor1 =  String("Sensor1:" + distancia1);
  delay(10);
```

```
digitalWrite(pingTrigger2, LOW);
delayMicroseconds(2);
digitalWrite(pingTrigger2, HIGH);
delayMicroseconds(5);
digitalWrite(pingTrigger2, LOW);
duration2 = pulseIn(Sensor2, HIGH);
cm2 = microsecondsToCentimeters(duration2);
distancia2 = String(cm2);
sensor2 =  String("Sensor2:" + distancia2);
delay(10);

digitalWrite(pingTrigger3, LOW);
delayMicroseconds(2);
digitalWrite(pingTrigger3, HIGH);
delayMicroseconds(5);
digitalWrite(pingTrigger3, LOW);
duration3 = pulseIn(Sensor3, HIGH);
cm3 = microsecondsToCentimeters(duration3);
distancia3 = String(cm3);
sensor3 =  String("Sensor3:" + distancia3);
delay(10);

digitalWrite(pingTrigger4, LOW);
delayMicroseconds(2);
digitalWrite(pingTrigger4, HIGH);
delayMicroseconds(5);
digitalWrite(pingTrigger4, LOW);
duration4 = pulseIn(Sensor4, HIGH);
cm4 = microsecondsToCentimeters(duration4);
distancia4 = String(cm4);
sensor4 =  String("Sensor4:" + distancia4);
delay(10);

digitalWrite(pingTrigger5, LOW);
delayMicroseconds(2);
digitalWrite(pingTrigger5, HIGH);
delayMicroseconds(5);
digitalWrite(pingTrigger5, LOW);
duration5 = pulseIn(Sensor5, HIGH);
cm5 = microsecondsToCentimeters(duration5);
distancia5 = String(cm5);
sensor5 =  String("Sensor5:" + distancia5);
delay(10);

digitalWrite(pingTrigger6, LOW);
delayMicroseconds(2);
digitalWrite(pingTrigger6, HIGH);
delayMicroseconds(5);
digitalWrite(pingTrigger6, LOW);
duration6 = pulseIn(Sensor6, HIGH);
cm6 = microsecondsToCentimeters(duration6);
```

```
  distancia6 = String(cm6);
  sensor6 =  String("Sensor6:" + distancia6);
  delay(10);

  digitalWrite(pingTrigger7, LOW);
  delayMicroseconds(2);
  digitalWrite(pingTrigger7, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingTrigger7, LOW);
  duration7 = pulseIn(Sensor7, HIGH);
  cm7 = microsecondsToCentimeters(duration7);
  distancia7 = String(cm7);
  sensor7 =  String("Sensor7:" + distancia7);
  delay(10);

 //converão para string e publicação no tópico ROS
  sensor1.toCharArray(s1, 22);
  sensor.data=s1;
  chatter1.publish(&sensor);
  nh.spinOnce();
  delay(10);
  sensor2.toCharArray(s2, 22);
  sensor.data=s2;
  chatter2.publish(&sensor);
  nh.spinOnce();
  delay(10);
  sensor3.toCharArray(s3, 22);
  sensor.data=s3;
  chatter3.publish(&sensor);
  nh.spinOnce();
  delay(10);
  chatter4.publish(&sensor);
  sensor4.toCharArray(s4, 22);
  sensor.data=s4;
  nh.spinOnce();
  delay(10);
  chatter5.publish(&sensor);
  sensor5.toCharArray(s5, 22);
  sensor.data=s5;
  nh.spinOnce();
  delay(10);
  chatter6.publish(&sensor);
  sensor6.toCharArray(s6, 22);
  sensor.data=s6;
  nh.spinOnce();
  delay(10);
  chatter7.publish(&sensor);
  sensor7.toCharArray(s7, 22);
  sensor.data=s7;
  nh.spinOnce();
  delay(10);
```

```
//desvio de obstaculos
  if((cm2>=50)and(cm3>=50)and(cm4>=50)) //sem obstaculos
  {
    frente();
  }
  if((cm1<20) and (cm7>10)) //obstáculo à direta
  {
    virar_esquerda();
  }
  if((cm5<20) and (cm6>10)) //obstáculo à esquerda
  {
    virar_direita();
  }
  if(((cm2<50)or(cm3<50)or(cm4<50)) and ((cm1<30)and(cm7>15)))
  {
    virar_esquerda();
  }else if (((cm2<50)or(cm3<50)or(cm4<50)) and ((cm5<30) and (cm6>15)))
  {
    virar_direita();
  }
 if((cm2<10)or(cm3<10)or(cm4<10)or(cm1<10)or(cm2<10)or(cm6<10)or(cm7<10))
 {
    parar();
 }
  if (comando_py==128) //recebeu sinal de parada via ROS
 {
    parar();
 }

  nh.spinOnce();
}
```

ANEXO C – Código para comando dos motores em python

```
#!/usr/bin/env python

from __future__ import print_function

import roslib; roslib.load_manifest('poter')
import rospy

from std_msgs.msg import Int16

import sys, select, termios, tty

msg = """
Comandos Poter via teclado
---------------------------
Movimentos:
        w
   a    s    d

Parada:
        p

Alterar velocidade:
c : aumenta
v : reduz

CTRL-C para sair
"""

erro = """
Velocidade limite
---------------------------

"""

moveBindings = {
        'p':(128),
        'w':(140),
        's':(120),
    }

speedBindings = {
        'c':(4),
        'v':(-4),
    }

turnRight = {
        'd':(118),
    }
```

```
turnLeft = {
        'a':(118),
    }

def getKey():
    tty.setraw(sys.stdin.fileno())
    select.select([sys.stdin], [], [], 0)
    key = sys.stdin.read(1)
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key

if __name__=="__main__":
    settings = termios.tcgetattr(sys.stdin)

    pub1 = rospy.Publisher('cmd_left_wheel', Int16, queue_size = 10)
    pub = rospy.Publisher('cmd_right_wheel', Int16, queue_size = 10)
    rospy.init_node('md49_keyboard')

    right = 140;
    left = 140;

    try:
        print(msg)
        while(1):
            key = getKey()

            if key in moveBindings.keys():
                right = moveBindings[key]
                left = moveBindings[key]

            elif key in speedBindings.keys():
                right = right + speedBindings[key]
                left = left + speedBindings[key]

            elif key in turnRight.keys():
                right = turnRight[key]

            elif key in turnLeft.keys():
                left = turnLeft[key]

            else:
                key == '\x03'
                break

            if right<152 and \
              right>110 and \
              left<152 and \
              left>110:
                  pub.publish(right)
                  pub1.publish(left)
            else:
```

```
                print(erro)
                print(msg)


        except Exception as e:
            print(e)

        finally:
            termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
```

ANEXO D – Código da página WEB em html

```
<!DOCTYPE html>
<html>
<title>Poter</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="css/w3.css">
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">
<head>
<!--
Integração entre JAVAScript e rosbridge
-->
<script type="text/javascript" src="js/eventemitter2.min.js"></script>
<script type="text/javascript" src="js/roslib.min.js"></script>
<script src="js/three.js"></script>
<script src="js/roslib.js"></script>
<script src="js/ros3d.js"></script>

<script>
// Inicia a conexão com o rosbridge na porta 8080
var rbServer = new ROSLIB.Ros({
    url : 'ws://192.168.0.17:8080'
 });
//tt

 // Conexão com o rosbridge
 rbServer.on('connection', function() {
     // mensagem de sucesso
     var fbDiv = document.getElementById('feedback');
     fbDiv.innerHTML += "<p>Connected to websocket server.</p>";
 });

rbServer.on('error', function(error) {
    // mensagem de erro
    var fbDiv = document.getElementById('feedback');
    fbDiv.innerHTML += "<p>Error connecting to websocket server.</p>";
});

// Chamada quando a conexão é encerrada
rbServer.on('close', function() {
    var fbDiv = document.getElementById('feedback');
    fbDiv.innerHTML += "<p>Connection to websocket server closed.</p>";
 });

// Cria um tópico
var cmdVelEsquerda = new ROSLIB.Topic({
    ros : rbServer,
    name : '/cmd_left_wheel',
    messageType : 'std_msgs/Int16'
});
```

```
var cmdVelDireita = new ROSLIB.Topic({
    ros : rbServer,
    name : '/cmd_right_wheel',
    messageType : 'std_msgs/Int16'
});

var listener = new ROSLIB.Topic({
    ros : ros,
    name : '/distancia',
    messageType : 'std_msgs/String'
  });

  listener.subscribe(function(message) {
    console.log('Sensores ultrassonicos ' + listener.name + ': ' + message.data);
    listener.unsubscribe();
  });

function init() {

    // Cria um objeto de visuzalição
    var viewer = new ROS3D.Viewer({
        divID : 'map',
        width : 800,
        height : 600,
        antialias : true
    });

    var gridClient = new ROS3D.OccupancyGridClient({
        ros : rbServer,
        rootObject : viewer.scene,
        continuous: true
    });
}

   function pubMessage() {
    /**
            Seta velocidade de parada na caixa-texto
    **/

    var rodaEsquerda = 128;
    var rodaDireita = 128;

    // adquire os valores inseridos na caixa-texto
    rodaEsquerda = Number(document.getElementById('RodaEsquerdaText').value);
    rodaDireita = Number(document.getElementById('RodaDireitaText').value);

    var cmdL = new ROSLIB.Message({
        data: rodaEsquerda
    });
```

```
    var cmdR = new ROSLIB.Message({
      data: rodaDireita
    });

    // publica as mensagens
    cmdVelEsquerda.publish(cmdL);
    cmdVelDireita.publish(cmdR);
  }
  </script>

  <style>
  body,h1,h2,h3,h4,h5,h6,ol {font-family: "Raleway", sans-serif}
  </style>

  <body class="w3-light-grey">

  <div class="w3-card-4 w3-margin w3-lightgray">
  <!-- Header -->
    <header class="w3-container w3-center w3-padding-32">
      <h1><b>POTER WEB</b></h1>
      <p>Interface de comunicaÃ§ao via <span class="w3-tag">ROS</span></p>
    </header>
  </div>

  <div class="w3-card-4 w3-margin w3-lightgray">

<h6>                &n
bsp;               &nb
sp;          Envie um comando ao Poter:</h6>
      <right>
        <form name="cmd_control">
          <span                                                                    class="w3-
opacity">               &nbs
p;                 
;                0:
velocidade maxima reversa</span>
          <p><span                                                                  class="w3-
opacity">               &nbs
p;                 
;                 
        128: velocidade nula</span>
          <p><span                                                                  class="w3-
opacity">               &nbs
p;                 
;                 
    255: velocidade maxima</span>
      </right>
  <div class="w3-col">
  <table align="center">
    <tr><td>Roda  esquerda</td><td><input  id="RodaEsquerdaText"  name="RodaEsquerdaText"  type="number"
value="140"></td></tr>
```

```
            <tr><td>Roda direita</td><td><input id="RodaDireitaText" name="RodaDireitaText" type="number" value="140">
</td></tr>
            <tr><td></td></tr>
          </table>
          </div>
            <button id="sendMsg" type="button" onclick="pubMessage()">Publicar mensagem</button>
            </form>
            <div id="feedback"></div>
          </div>
          <div align="center">
          <div class="w3-card-4 w3-margin w3-gray">
            <h3>Mapeamento com RPLidar</h3>
              <body onload="init()">
                <p>Para vizualizar execute os comandos no Odroid via terminal:
                <p><span class="w3-opacity">roslaunch rplidar_ros rplidar.launch</span>
                <p><span class="w3-opacity">roslaunch hector_slam_launch tutorial.launch</span>
                <p><span class="w3-opacity">roslaunch rosbridge_server rosbridge_websocket.launch</span>
          <div id="map"></div>
          </div>
          </div>
          </body>
          </html>
```