UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA

DHIANCARLO ALMEIDA MARTINS

PROJETO DO ROBÔ DIDÁTICO ATTABOT 2.0

Uberlândia
Agosto de 2022

DHIANCARLO ALMEIDA MARTINS

PROJETO DO ROBÔ DIDÁTICO ATTABOT 2.0

Monografia apresentada ao Programa de Graduação em Engenharia Mecatrônica da Universidade Federal de Uberlândia, como parte dos requisitos para obtenção do título de Engenheiro Mecatrônico.

Orientador: José Jean-Paul Zanlucchi de Souza Tavares

Uberlândia
Agosto de 2022

AGRADECIMENTOS

Aos meus pais que sempre demonstraram apoio, acreditaram, investiram e foram muito pacientes na minha educação.

Ao Professor José Jean-Paul Zanlucchi de Souza Tavares pela oportunidade de desenvolver este projeto e as ótimas palavras reconfortantes que sempre me faziam desistir de desistir.

A todos que inconscientemente com suas palavras me mostraram que eu consigo e me motivaram a continuar.

RESUMO

Com o desenvolvimento acelerado da tecnologia, em especial da robótica e a visão de que no futuro haverá um crescente uso de robôs nos mais diversos trabalhos, vem a necessidade de preparação de profissionais para programação, controle e manutenção desses dispositivos. Uma das formas de difusão desse conhecimento é por meio do aprendizado prático na formação dos recursos humanos.

Este trabalho contém o projeto de um robô didático com a visão de aplicação em unidades de ensino médio, técnico e superior com o intuito de introduzir mais conhecimento prático aos alunos que um dia ocuparão esses cargos da robótica, conhecimentos que vão desde sensores, integração de componentes e sistemas de simulação como o ROS ("Robot Operating System"). O resultado é o projeto de um protótipo denominado Attabot 2.0.

Palavras-chaves: Robô didático, Protótipo de robô, Sistema Operacional para Robôs

ABSTRACT

With the accelerated development of technology, especially robotics and the vision that in the future there will be an increasing use of robots in the most diverse jobs, comes the need to prepare professionals for programming, control and maintenance of these devices. One of the ways of disseminating this knowledge is through practical learning in the training of human resources.

This work contains the project of a didactic robot with the vision of application in high, technical and higher education units in order to introduce more practical knowledge to students who will one day occupy these robotics positions, knowledge ranging from sensors, integration of components and simulation systems such as ROS ("Robot Operating System"). The result is the design of a prototype called Attabot 2.0.

Keywords: Didactic robot, Robot prototype, Robot Operating System

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ROS	*Robot Operating System*
IDE	*Integrated Development Environment*
ARM	*Advanced RISC Machine*
PWM	*Pulse Width Modulation*
UART	*Universal Asynchronous Receiver/Transmitter*
I2C	*Inter-Integrated Circuit*
SPI	*Serial Peripheral Interface*
MHz	Megahertz
KB	Kilobyte
SRAM	*Static Random Acess Memory*
V	Volt
I/O	*Input/Output*
RX	*Receiver*
TX	*Transmitter*
SDA	*Serial Data*
SCL	*Serial Clock*
MQTT	*Message Queue Telemetry Transport*
IoT	*Internet of Things*
TCP	*Transmission Control Protocol*
IP	*Internet Protocol*
DC	*Direct Current*
mm	milimetro
RPM	Revolução por Minuto
PPR	Pulso por Rotação
IMU	*Inertial Measurement Unit*
DMP	*Digital Motion Processor*
VGA	*Video Graphics Array*
fps	*frames per second*
LDR	*Light Dependent Resistor*
Li-Ion	íon de lítio
mAh	miliampere-hora
CAD	*Computer Aided Design*
URDF	*Unified Robot Description Format*
XML	*Extensible Markup Language*
RFID	*Radio-Frequency Identification*
CMOS	*Complementary metal-oxid-semiconductor*
GHz	Gigahertz
BLE	*Bluetooth Low Energy*
LED	*Light-emitting diode*

Sumário

1 INTRODUÇÃO

A robótica é a ciência que estuda as tecnologias associadas a concepção e construção de robôs. Os robôs são mecanismos automáticos que utilizam de circuitos integrados para realizarem atividades e movimentos simples ou complexos em áreas desde a produção industrial, medicina e até atividades domésticas [Portal da indústria]. Para que alguém desenvolva um robô, esta pessoa deve dotar de conhecimentos multidisciplinares como matemática, física, mecânica, eletrônica, computação e outros. No futuro, as profissões que envolvem tecnologia lidarão diretamente com alguma face da robótica, por isso é preciso preparar os profissionais para esse mundo tecnológico. Nesse sentido, a robótica se destaca como uma formação essencial. Muito além do conhecimento técnico, a robótica ajuda a desenvolver habilidades como raciocínio lógico, criatividade, resolução de problemas e trabalho em equipe [Pio, 2006]. Por esses motivos, a robótica vem aparecendo cada vez mais como disciplina nos currículos escolares.

Essa necessidade por profissionais mais capacitados na área exige que a robótica venha a ser difundida antes mesmo que o aluno ingresse no ensino superior. O estudo da robótica exige prática para além da teoria, um aprendizado baseado na experimentação de sensores e movimentos torna o aluno capaz de solucionar por si só eventuais problemas que venha a encontrar [Simões, 2006].

No ensino superior, a falta de atividades práticas dedicadas ao manuseio de um robô, mesmo em cursos como Engenharia Mecatrônica, desmotiva o aluno quanto à sua formação. Neste contexto, cabe a criação de um robô didático capaz de introduzir o conhecimento da robótica em iniciantes e aprimorar em veteranos através da experimentação. No caso da robótica educacional, a mesma deve ter ênfase na utilização de hardwares de baixo custo e softwares gratuitos.

A partir da constatação da realidade e necessidade acima mencionada, o robô aqui apresentado foi estudado e projetado afim de se tornar aplicável em laboratórios de ensino com o intuito de apresentar aos estudantes atividades práticas que ajudam a desenvolver habilidades cada vez mais requisitadas.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho é o projeto de um protótipo de robô móvel denominado Attabot 2.0, com o intuito de servir de referência didática no ensino da robótica em cursos de Engenharia Mecatrônica, auxiliar cursos técnicos na montagem desse tipo de dispositivo e servir de estímulo para cursos de ensino médio.

1.1.2 Objetivos Específicos

Os objetivos específicos são:

- Especificar o protótipo Attabot 2.0;
- Realizar testes funcionais;
- Integrar a solução ao ROS (*Robot Operating System*).

1.2 Justificativa

Ao longo de todo o curso de Engenharia Mecatrônica o conhecimento adquirido nas seguintes matérias foram essenciais para a realização deste trabalho: Algoritmos e Programação de Computadores, Projeto Assistido por Computador, Eletrônica Básica para Mecatrônica, Sistemas Operacionais, Instrumentação, Sistemas Digitais para Mecatrônica, Redes Industriais, Processamento Digitais de Sinais e matérias de Controle de Sistemas.

Acreditando na carência de ferramentas que contribuam para o ensino da mecatrônica para alunos da universidade, cursos técnicos e também para alunos do ensino médio, de forma a acompanhar o avanço atual da tecnologia destinada à robótica, existe a necessidade de projetos robóticos didáticos integrados com ferramentas como o *ROS*, contribuindo para a democratização da ciência.

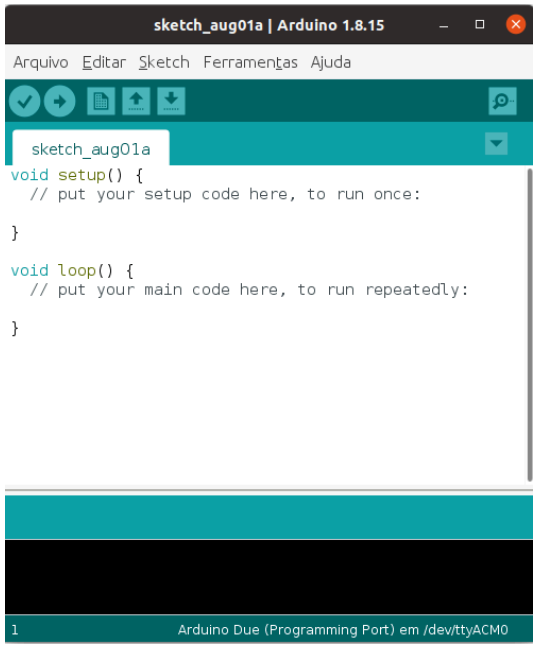
2 FUNDAMENTAÇÃO TEÓRICA

2.1 Microcontroladores

O Arduino é um microcontrolador indicado para projetos eletrônicos e elétricos de controle de hardware, como sensores e motores pelo seu ótimo custo benefício. Sua programação é composta basicamente da linguagem C podendo-se fazer uso também de C++. Aqui é feito o uso de três módulos Arduinos, o principal é o Arduino Due responsável pela integração de todos os sensores, os outros dois são Arduinos Nano responsáveis individualmente pela integração com os dois motores.

Toda a programação dos microcontroladores é feita através da IDE ou *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado) própria do Arduino que trata-se de uma ferramenta de desenvolvimento para editar o código, executar um script, debugar e compilar em um único ambiente e através da enorme comunidade existente em todo o mundo possibilita o uso de uma imensa variedade de bibliotecas específicas para cada tipo de sensores que aqui será utilizado.

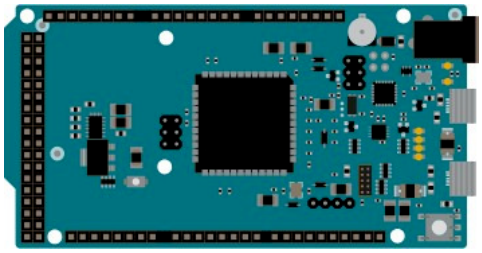
Figura 1: Arduino IDE



2.1.1 Arduino Due

O Due é o primeiro módulo Arduino microcontrolador baseado em núcleo ARM de 32 bits. Possui 54 pinos de entrada/saída digital (dos quais 12 podem ser usados como saídas PWM), 12 entradas analógicas, 4 UARTs (portas seriais de hardware), I2C, SPI, um processador AT91SAM3X8E com clock de 84MHz, memória de 96KB SRAM e 512KB flash, opera com alimentação nominal de 7-12V e 3.3V em suas portas I/O. Ele possui também um módulo QEI (*Quadrature Encoder Interface*) de 2 entradas para acompanhamento de posição e velocidade angular de motores com encoder)

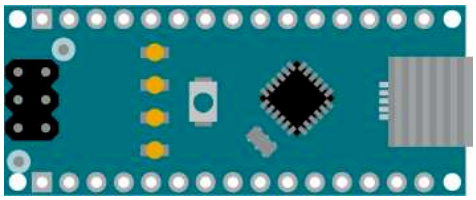
Figura 2: Arduino Due



2.1.2 Arduino Nano

Com 45mm de comprimento e 18mm de largura o Nano é o menor dos módulos da família Arduino. Feito para uso em protoboard é baseado em núcleo AVR de 8 bits, possui 14 pinos de entrada/saída digital (dos quais 6 podem ser usados como saída PWM), 8 entradas analógicas, 1 UART (porta serial de hardware), I2C, SPI, um processador ATmega328 com clock de 16MHz, memória de 2KB SRAM e 32KB flash, opera com alimentação nominal de 7-12V e 5V em suas portas I/O.

Figura 3: Arduino Nano



2.2 Sistema de comunicação

2.2.1 UART

Em português, UART significa Transmissor/Receptor Universal Assíncrono, como o próprio nome diz é uma comunicação assíncrona, ou seja, não faz uso de um sinal de clock. Essa comunicação é full duplex, realizada ao mesmo tempo de um dispositivo para o outro pois os dados trafegam por meios diferentes fazendo uso das portas RX e TX do Arduino.

2.2.2 I2C

O I2C ou Inter-Integrated Circuit (Circuito Inter-Integrado) utiliza dois canais para comunicação sendo um para transmissão e recepção de dados (SDA) e outro para sincronização (SCL), ou seja, é um tipo de comunicação síncrona. Para haver comunicação por meio desse protocolo deve existir pelo menos um dispositivo "mestre", nesse caso trata-se do microcontrolador principal Arduino Due, e outro "escravo" que serão todos os sensores e também os Arduinos Nano, sendo possível a ligação de vários escravos em um único mestre. O mestre é responsável pela coordenação da comunicação enquanto os escravos respondem quando são chamados. Cada escravo possui um endereço individual de 7 bits que é utilizado pelo mestre para se comunicar individualmente com eles, esse endereço normalmente é escrito em hexadecimal.

2.2.3 SPI

O SPI ou Serial Peripheral Interface (Interface Periférica Serial) assim como o I2C, é um tipo de comunicação síncrona entre dispositivos definidos com um único mestre e vários escravos. Constitui-se de no mínimo quatro canais diferentes:

- MOSI (Master Out Slave In): Canal utilizado pelo mestre para enviar dados aos escravos;
- MISO (Master In Slave Out): Utilizado pelo escravo para enviar dados ao mestre;
- SCLK (Serial Clock): Canal para sincronizar a transmissão de dados;
- SS (Slave Select): Utilizado para selecionar o escravo ao qual o mestre vai se comunicar.

Sendo assim, deve haver um canal SS para cada escravo se comunicando via SPI com o mesmo mestre, diferentemente do I2C que utiliza o mesmo canal mudando apenas o endereço ao qual deseja se comunicar.

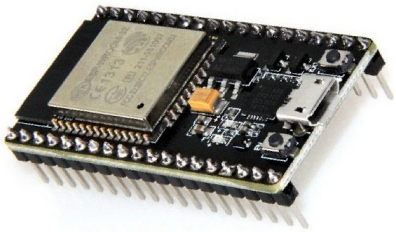
2.2.4 MQTT

O MQTT (Message Queue Telemetry Transport) é um protocolo de comunicação entre máquinas amplamente utilizado em IoT que permite o envio de mensagens e comandos entre dispositivos por meio de TCP/IP.

2.3 Módulo WiFi ESP32

O ESP32 é um dispositivo IoT de 32 bits com suporte à rede WiFi, Bluetooth e memória flash integradas podendo ser programado de forma independente, sem a necessidade de um microcontrolador como o Arduino. Através dele e do protocolo MQTT é possível o envio e leitura de sinais pela rede sem fio de um robô para a nuvem, permitindo assim a comunicação baseada em feromônio de formiga.

Figura 4: ESP32



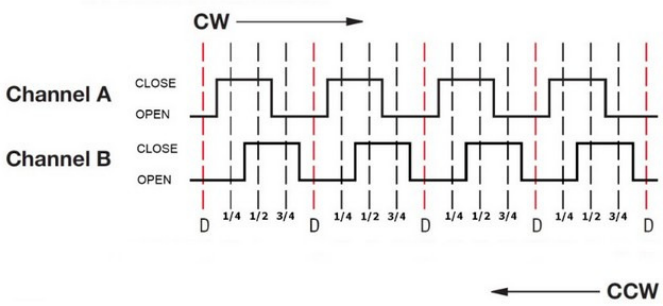
2.4 Motores com encoder

Os dois motores DC utilizados no projeto são o N20 com encoder integrado, apesar do tamanho de 30,5 x 18,5mm possuem um sistema de engrenagens que entregam rotações de até 150 RPM podendo ser ajustado de acordo com as necessidades do projeto através do encoder. Sua alimentação varia de 3 a 12V e necessitam de um driver para controle dos mesmos, vistos no item 2.4.1.

Figura 5: Motor N20 com encoder

Um encoder de quadratura é um encoder incremental com 2 canais de saída A e B defasados para detecção da velocidade e direção do movimento. Cada canal fornece um número específico de pulsos por rotação (PPR) igualmente espaçados e a direção do movimento é detectada pela relação de fase de um canal ao outro (adiantado ou atrasado). O disco de código dentro deste encoder possui duas faixas, ou trilhos que se ligam diretamente ao canal A e B cada um. A separação entre cada trilho reflete um sinal de saída com 90° de diferença entre cada sinal para cada fase como mostra a Figura 6.

Figura 6: Funcionamento do Encoder

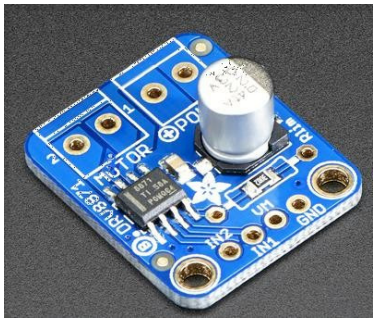


2.4.1 Driver para motor DC

Um driver para motor também conhecido como ponte H é um circuito que serve para variar o sentido da corrente de saída, bem como controlar sua potência, sendo assim, através desse driver define-se em qual sentido o motor irá girar sem sobrecarga.

O DRV8871 possui controle PWM que refere-se ao conceito de pulsar rapidamente um sinal digital em um condutor, através disso e do uso do encoder é possível definir a velocidade de giro do motor.

Figura 7: Driver DRV8871

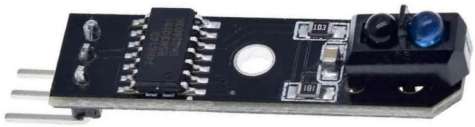


2.5 Sensores

2.5.1 Sensor óptico

O sensor óptico é constituído de um emissor (led infravermelho) e um receptor (fototransistor). O emissor emite uma faixa de luz que quando é refletida em uma superfície e é captada pelo receptor causa uma queda de tensão entre o coletor e o emissor dependendo da intensidade do sinal. É comumente utilizado em projetos de seguidores de linha, no caso do projeto Attabot 2.0 ele atua como um sensor de colisão devido ao seu alcance limitado de até 5 mm para um bom funcionamento, possuindo assim 6 desses sensores ao longo de sua circunferência para identificar possíveis colisões em várias direções.

Figura 8: TCRT5000



2.5.2 Módulo acelerômetro e giroscópio

Conhecido também como IMU ou Unidade de Medição Inercial possui 9 eixos que combina um giroscópio, um acelerometro e um magnetômetro de 3 eixos cada além de um processador de movimento digital ou DMP. Serve para estimar a posição do robô no espaço, bem como sua direção facilitando a tomada de decisões. Sua comunicação é feita através do protocolo I2C.

Figura 9: IMU MPU9250

2.5.3 Sensor de distância laser

Nosso sensor de distância possui alcance de até 2 metros com alta precisão. Em seu funcionamento o sensor envia um laser invisível a olho nu e ao ser refletido de volta captura a distância percorrida pelo laser através do tempo de resposta. Seu laser possui um ângulo de abertura de 25° e sua comunicação é feita via protocolo I2C.

Figura 10: Sensor de distância VL53L0X

2.5.4 Módulo câmera VGA

Trata-se de uma camera integrada a um módulo que permite a captura e armazenamento de imagens com uma taxa de atualização de até 30 fps com resolução de 640 x 480 pixels. Possui alta sensibilidade em ambientes com pouca luz, é compatível com I2C, possui controle de qualidade de imagem como saturação, matriz, gama e nitidez. Através da câmera e de bibliotecas específicas na programação é possível a identificação de objetos e/ou cores ao custo de sua taxa de transmissão e resolução.

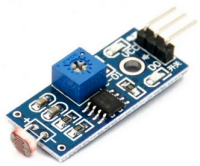
Figura 11: Módulo câmera VGA OV7670

2.5.5 Sensor de luminosidade

O módulo LDR é um sensor de luz baseado em um foto resistor que mede a intensidade da luz ambiente através da variação de sua resistência interna. Nesse

caso, pode ser utilizado para identificar pontos de alta luminosidade no ambiente.

Figura 12: Módulo LDR



2.6 Baterias

As células de baterias Li-Ion 18650 possuem individualmente carga máxima de 3800mAh, tensão de 3,7V e são recarregáveis. Para atender aos requisitos do projeto duas baterias foram designadas para uso dos motores e uma bateria para os demais sensores, havendo a necessidade de uso de cases especiais para uso das mesmas.

2.7 Software CAD

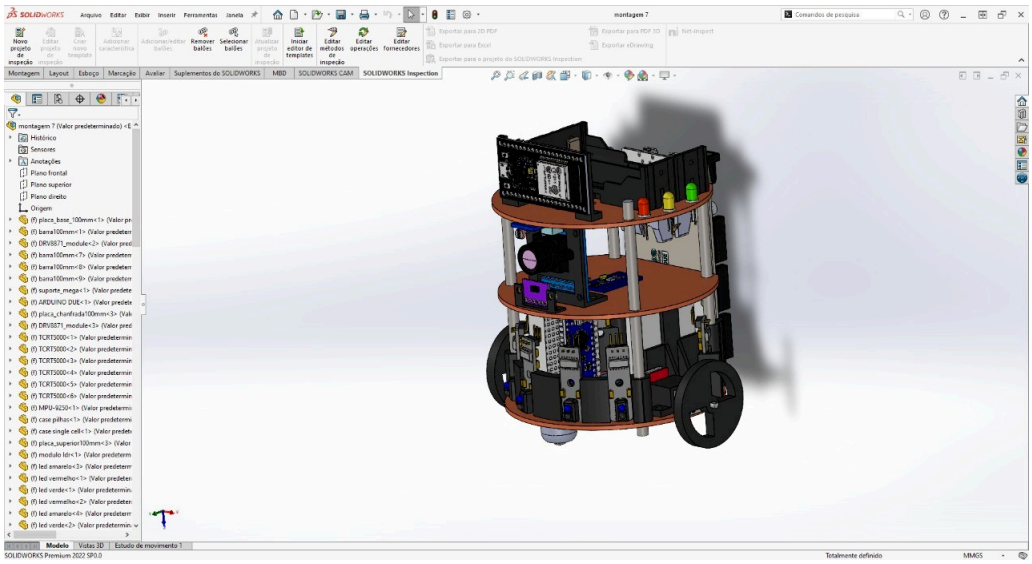
De forma geral, todos modelos tridimensionais dos sensores, motores e cases de baterias foram obtidos na plataforma *GrabCAD* havendo apenas a necessidade de ajustes de dimensões com os componentes reais. Partes como as bases do robô, rodas e suportes dos sensores e também a junção de todos os itens foram feitas com uso do software *SolidWorks*.

O *SolidWorks* baseia-se em computação paramétrica, criando formas tridimensionais a partir de operações geométricas elementares. No ambiente do programa, a criação de um sólido começa com a definição de um sketch 2D que depois é transformado num modelo tridimensional. Através de um amplo leque de funcionalidades o *SolidWorks* atende a demanda de projetos de engenharia de diversas áreas que se baseiam em criação de sólidos com geometrias as vezes complexas, possibilitando a criação de peças, montagem de componentes, simulação de movimentos, análise de material, entre outros.

2.8 Robot Operating System (*ROS*)

O *ROS* fornece bibliotecas e ferramentas para ajudar os desenvolvedores de software a criar aplicativos de robôs. Ele fornece abstração de hardware, drivers de dispositivo, bibliotecas, visualizadores, transmissão de mensagens, gerenciamento de pacotes e muito mais. O *ROS* possui licença BSD de código aberto e uma

Figura 13: *SolidWorks*



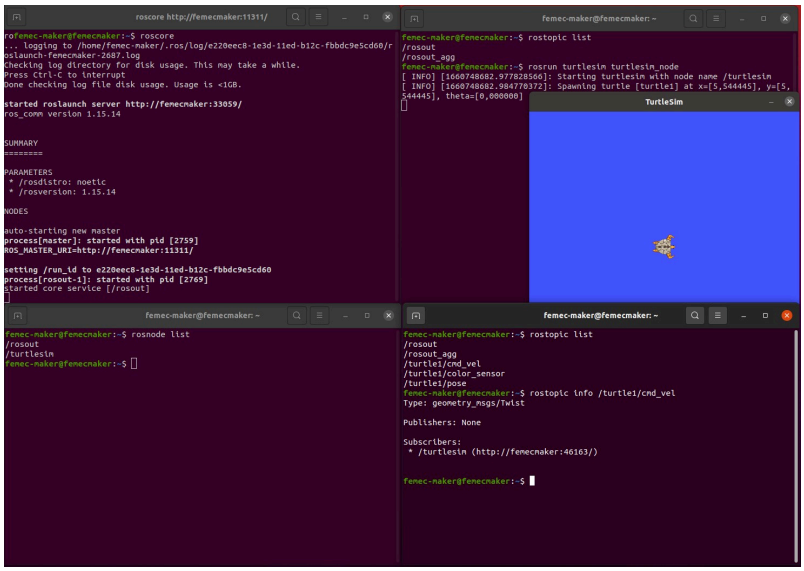
enorme gama de tutoriais e exemplos pré desenvolvidos pela comunidade para serem usados.

Na Figura 14 vê-se um uso típico de *ROS*, deve-se primeiro iniciá-lo em um terminal através do comando *roscore* em qualquer que seja o projeto, no terminal à direita vê-se a inicialização do nó *turtlesim*, um nó *ROS* é basicamente um processo que realiza computação, é um programa executável dentro do seu aplicativo, os nós são combinados em um gráfico e se comunicam usando tópicos, serviços e ações do *ROS*. O *turtlesim* é uma ferramenta feita para ensinar o uso do *ROS* e seus pacotes, através da janela em azul é possível ver e entender conceitos importantes como movimentos no plano. Nos terminais inferiores vê-se os nós atualmente ativos e os tópicos associados a eles, os tópicos possuem um tipo de mensagem que podem ser chamadas durante a programação dos códigos a serem usados no projeto, como por exemplo o tópico */turtle1/cmd_vel* que possui mensagens do tipo *geometry_msgs/Twist* que quando usado na programação controla o movimento realizado pela tartaruga de exemplo.

Juntamente com o *ROS* existem duas ferramentas muito importantes que permitem a visualização do robô e seus sensores, *Rviz* e *Gazebo*. A primeira permite que você visualize o estado de um robô usando dados de sensores para tentar criar uma representação precisa do que está acontecendo no ambiente. A segunda é um simulador que pode até calcular o impacto de forças como a gravidade, mostrando aproximadamente como o robô se comportaria em um ambiente físico no mundo real.

A diferença entre as duas ferramentas pode ser resumida no seguinte trecho

Figura 14: *ROS*



de Morgan Quigley (um dos desenvolvedores originais do ROS) em seu livro *Programming Robots with ROS* que quando traduzido fica: ”Rviz mostra o que o robô pensa que está acontecendo, enquanto o Gazebo mostra o que realmente está acontecendo.”

2.9 Exportador URDF

O URDF exporter é uma extensão para o *SolidWorks* que permite a exportação de peças e montagens para um arquivo URDF. Um arquivo URDF (Unified Robot Description Format) ou Formato de Descrição do Robô Unificado é uma especificação XML para descrever robôs ao qual é lida e interpretada pelo *ROS*. Essa especificação cobre a cinemática, dinâmica, representação visual e modelo de colisão do robô e sua descrição consiste de elos e juntas.

3 METODOLOGIA

Do ponto de vista da engenharia, a robótica móvel requer habilidades de várias disciplinas como mecânica, eletrônica, gerenciamento de energia, ciência da computação, processamento de sinais e controle e automação, a combinação de todas essas áreas faz com que um robô dedicado ao ensino seja uma excelente oportunidade prática de aprendizagem.

A execução deste projeto de fim de curso procede o projeto Attabot (FERREIRA, Marco Vinícius Muniz, 2020) que utilizava de tags RFID para emulação de feromônio de formiga em um enxame de robôs. Aqui, para o mesmo princípio do feromônio de formiga a comunicação entre o enxame é feita através de um servidor em nuvem a partir do ESP32 utilizando o protocolo MQTT. Dessa forma, a medida em que um robô coleta dados de uma determinada posição do ambiente os demais robôs que compoem o enxame possuem acesso a estes dados.

Além da primeira versão do Attabot desenvolvido nesta universidade, este projeto também tem como base o e-puck da École Polytechnique Fédérale de Lausanne (EPFL).

3.1 Revisão Bibliográfica

3.1.1 Attabot

O primeiro Attabot desenvolvido como tese de doutorado utilizava da tecnologia de identificação por rádio frequência para tomada de decisão. O RFID é o responsável por essa comunicação via ondas de rádio para troca de dados entre um leitor e uma etiqueta eletrônica (também chamada de tag). A ideia é que o robô que possui o leitor ao passar por alguma das tags estrategicamente posicionadas no ambiente recebe o sinal da tag e assim atualiza seu status e muda seu comportamento.

Além do leitor RFID, o primeiro Attabot possui dois sensores ultrassônicos HC-SR04, sensores de linha 4-way, módulo relé, motores DC 6V, ponte H L9110S e baterias Li-ion 4,2V tudo isso conectado a uma placa Arduino Mega.

Figura 15: Tag e leitor RFID

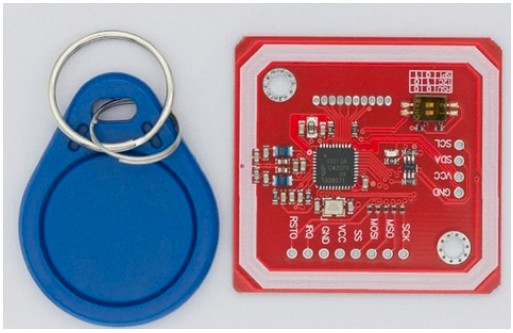
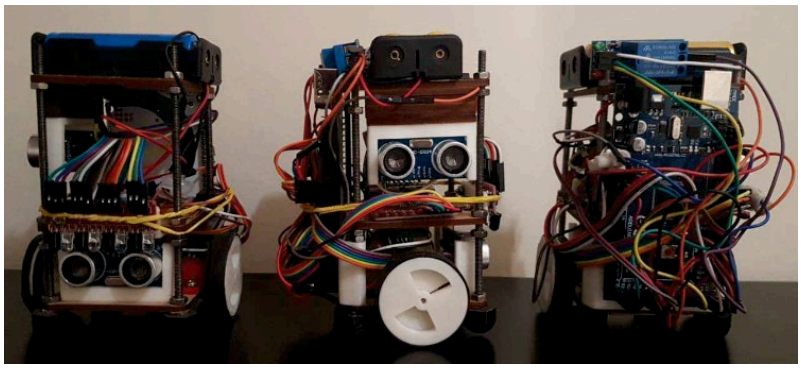


Figura 16: Primeira versão do Attabot



3.1.2 e-puck

O e-puck, originalmente desenvolvido por Francesco Mondada e Michael Bonani na École Polytechnique Fédérale de Lausanne em 2009 na Suíça teve como objetivo auxiliar o ensino de engenharia no local. A primeira versão é composta por diferentes tipos de sensores como audio, video, distância de objetos e gravidade e sua comunicação se dá com ou sem fio.

O microcontrolador dsPIC usado possui 16 bits e uma unidade de processamento digital de sinais, oito sensores infravermelhos são dispostos ao longo de sua circunferência afim de detectar obstáculos e colisões, um acelerômetro 3D provê um vetor de aceleração que é usado para medir a inclinação e a aceleração produzida pelo movimento, três microfones capturam o som através de triangulação, uma câmera CMOS com resoluçao de 640 x 480 pixels instalada na parte frontal permite o processamento de imagens porém devido à taxa de aquisição limitada pelo tamanho da memória do dsPIC o e-puck pode adquirir imagens de 40 x 40 pixels a 4 frames por segundo (8 se estiver em escala de cinza). Essa limitação mostra aos estudantes o impacto de grandes larguras de banda em sensores como câmeras.

Além dos sensores descritos, o e-puck possui dois motores de passo, um speaker e diversos leds em seu contorno para comunicação visual com o usuário e/ou outros e-pucks através da câmera.

Figura 17: e-puck

3.2 Análise de requisitos

Afim de atender à demanda por um robô de bancada que possa ser usado para o ensino de engenharia no ambiente da universidade, os requisitos para este projeto são:

Tabela 1: Requisitos

	Requisito
R01	O robô deve ser capaz de movimentar de forma autônoma
R02	Interface simples e acessível: Deve ser didático ao aluno e atender ao quesito “Faça Você Mesmo”
R03	Tamanho compacto: Deve ser de fácil transporte e manuseio em bancada
R04	Baixo custo
R05	Open source e disponível na internet
R06	Controle de dois motores com encoder para posição e velocidade angulares
R07	Deve ter comunicação em nuvem
R08	Deve ter um sistema anti colisão
R09	O robô deve possuir um sistema de medição de distância de objetos
R10	Possuir um sistema de reconhecimento de imagem;
R11	Deve fazer detecção de pontos luminosos
R12	O robô deve ser capaz de fazer o mapeamento de posição e trajetória
R13	Deve possuir um processador de 32 bits

Assim, através da análise de componentes e discussão com professores com amplo conhecimento no assunto chegou-se aos itens dispostos no capítulo 2.

As dimensões iniciais definidas no projeto do robô são 100mm de altura e 100mm de circunferência com flexibilização de 50%.

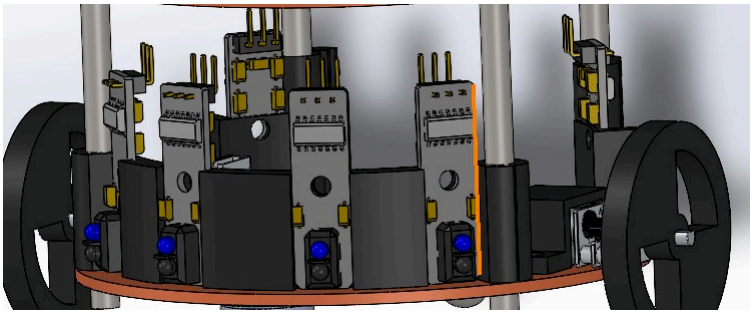
4 DESENVOLVIMENTO

Através de reuniões com membros do projeto Femec Maker chegou-se às análises de requisitos descritas no item 3.2, englobando o projeto e-puck (3.1.2) e o primeiro Attabot (3.1.1) concluiu-se que os sensores, atuadores e microcontroladores descritos no capítulo 2 atenderiam às necessidades do projeto descritas a seguir.

Havendo a necessidade de integração em nuvem, o responsável pela mesma é o ESP32 (item 2.3). Usado também em outros projetos do Femec Maker, o ESP32 foi escolhido pelo seu amplo uso em aplicações IoT por ser um módulo de alta performance para aplicações envolvendo comunicação sem fio. Seu Wifi de 2.4 GHz é o responsável pela comunicação principal e futuramente seu Bluetooth BLE 4.2 será usado para comunicação entre robôs.

Afim de detectar possíveis colisões, partindo do projeto e-puck que utiliza de sensores TCRT1000, os sensores infravermelhos do item 2.5.1 foram escolhidos por serem da mesma família e já virem integrados em módulos tornando mais fácil sua integração e uso. Os 6 sensores são estrategicamente posicionados no robô a fim de identificar estas colisões em várias direções como mostra a Figura 18.

Figura 18: Disposicão dos Sensores de colisão



Da mesma forma, o sensor de distância do item 2.5.3 também é usado no e-puck e pela sua alta precisão, alcance de até 2 metros e facilidade de comunicação I2C atende bem à nossa necessidade de medir distâncias de objetos e obstáculos.

O IMU (item 2.5.2) já usado algumas vezes no laboratório possui interface I2C e SPI e a ideia da integração do mesmo neste robô para estimar a posição no espaço e sua direção também vem do uso de um IMU no e-puck com o mesmo propósito.

A câmera (item 2.5.4) foi colocada no projeto com o intuito de identificação por imagens, a partir daí é possível seu uso para identificação de objetos, cores (como nos leds) e um possível seguidor de linha. O desenvolvimento do software para estes reconhecimentos não faz parte do escopo do projeto.

O uso do módulo LDR do item 2.5.5 vem para indicar fontes com muita luz no ambiente, o sensor futuramente será utilizado na implementação de algoritmos de definição de trajetória baseado na iluminação.

Alguns leds foram dispostos no topo do robô com o intuito de serem usados como uma comunicação visual sendo possível a identificação de algum status do robô através da combinaçao de leds que estão ativos em determinado momento.

Os dois motores DC responsáveis pelo movimento do robô foram usados no primeiro Attabot com a diferença de que aqui eles possuem individualmente um encoder de quadratura com a finalidade de que seja possível o controle de velocidade e posição angular dos mesmos em malha fechada.

Os drivers escolhidos (item 2.4.1), como já dito, possuem controle PWM e através deles e da leitura dos encoders o controle do giro dos motores pode ser feito.

Para a integração de todos estes sensores e motores inicialmente foi pensado no uso do microcontrolador Arduino Mega que possui um processador de 8 bits, a ideia do mesmo foi substituída pelo Arduino Due por possuir um processador de 32 bits que atende aos requisitos do projeto. Em geral, passar de microcontroladores de 8 para 32 bits significa que terá menos restrições de recursos, principalmente memória e a largura de registros usados para operações aritméticas e lógicas. Sendo assim, o novo microcontrolador escolhido casa perfeitamente com o ESP32 por exemplo, otimizando assim a troca de dados entre as placas.

Um obstáculo encontrado no projeto refere-se ao uso dos encoders. Para que seja possível a leitura mútua de ambos cada motor recebeu seu próprio microcontrolador Arduino Nano, caso contrário, se os dois encoders estivessem conectados ao microcontrolador principal a leitura seria feita individualmente, o que ocasionaria uma defasagem no tempo de resposta e por consequência uma diferença de giro entre os motores.

Cabe então definir a alimentação de todo o circuito. Como os motores exigem demasiada energia para funcionamento contínuo e operam com tensão de 3 a 12V, duas baterias foram dedicadas exclusivamente para os mesmos enquanto uma única bateria fica responsável pela alimentação do circuito restante. Aqui, cabe definir um circuito regulador de tensão para que a alimentação não seja maior do que o suportado pelos equipamentos.

Com todos os itens definidos, o diagrama resultante do Attabot 2.0 nos mostra as conexões e a comunicação entre os dispositivos pelo sentido das setas (Figura 19).

A lista de componentes do Attabot 2.0 encontra-se na Tabela 1. Os valores unitários e com frete referem-se à cotação de março de 2022, apenas os motores e os drivers necessitaram de importação.

Figura 19: Diagrama esquemático Attabot 2.0

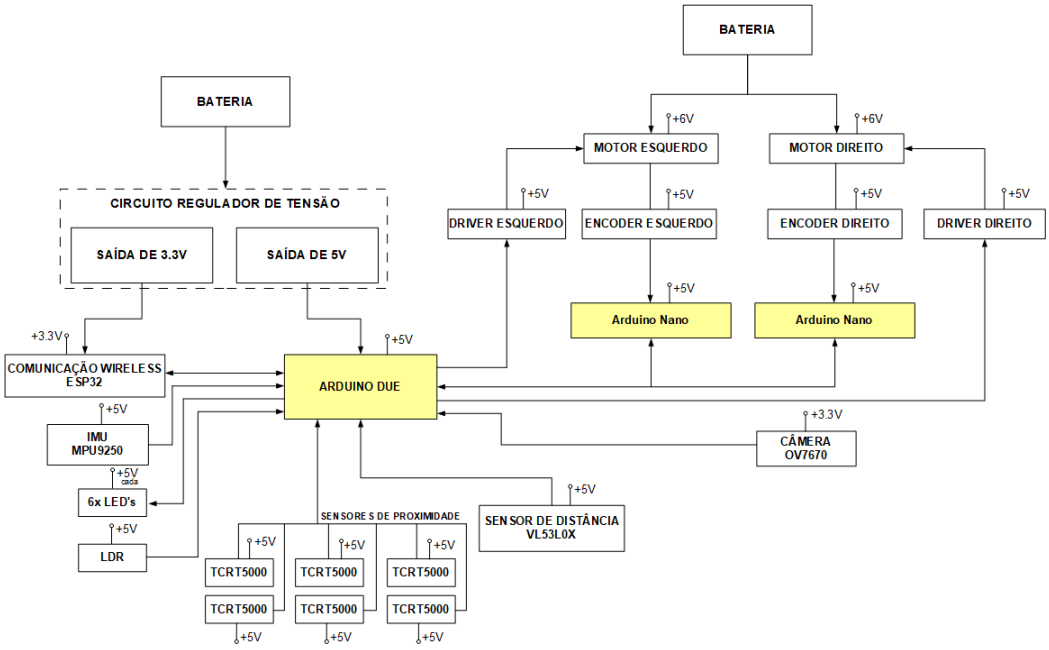


Tabela 2: Lista de componentes Attabot 2.0 - preços de março de 2022

Qtd	Item	Nome	Valor Unit.	Total com Frete
1	Módulo câmera	OV7670	R$99,99	R$99,99
1	Sensor de distância	VL53L0X	R$43,89	R$69,19
6	Sensor óptico	TCRT5000	R$7,90	R$59,76
1	Acelerômetro e giroscópio	MPU9250	R$67,20	R$90,46
1	Sensor de luminosidade	Módulo LDR	R$8,01	R$27,69
6	LED 5mm		R$0,25	R$1,50
2	Driver para motor DC	DRV8871	R$17,26	R$56,57
2	Motor com encoder	N20	R$25,29	R$82,86
2	Mini protoboard		R$5,90	R$23,93
1	Módulo WiFi	ESP32	R$69,73	R$86,03
1	Case dupla bateria 18650		R$14,00	R$39,60
1	Case unitária bateria 18650		R$16,90	R$41,30
3	Baterias 18650		R$16,90	R$75,80
1	Microcontrolador mestre	Arduino Due	R$197,38	R$197,38
2	Microcontrolador escravo	Arduino Nano	R$42,00	R$97,87
				TOTAL R$1049,93

4.1 Modelagem 3D

O próximo passo é a modelagem tridimensional do robô. Utilizando a plataforma *GrabCAD* foi obtido de forma livre os arquivos compatíveis com *SolidWorks* de cada um dos itens definidos para o projeto.

Definiu-se a montagem em três níveis, no nível inferior encontra-se as rodas, os motores, drivers e microcontroladores dedicados, uma terceira roda de apoio na parte traseira, a base contendo o microcontrolador principal Arduino Due que se prolonga por toda a altura do robô e os seis sensores de colisão no nível das rodas. O nível médio contém a camera, o sensor de distância e o IMU. No nível superior estão os leds, as cases das baterias, o LDR e principalmente o ESP32.

A medida em que se dispunha os componentes nas bases foram ocorrendo problemas de espaço, os fios de ligações que viriam a ser colocados no protótipo não foram levados em consideração. Para que todos os itens da parte inferior coubessem, os Arduinos Nano foram dispostos na posição vertical conectados a mini protoboards, o que também resolveu a questão de como as ligações seriam feitas no mesmo. Como mostrado na Figura 18, foi necessário a criação de um suporte para manter os sensores de colisão nas posicões. O suporte usado para o Arduino Due é o mesmo usado no primeiro Attabot com uma modificação para comportar mais dois sensores de colisão.

A base do meio não houve problema com alocação de componentes, já a parte superior após algumas tentativas foi conseguido um posicionamento que comportasse as cases de baterias e o ESP32 que necessita ficar na parte mais superior do robô para que haja o mínimo de interferência na comunicação wireless entre robô e nuvem.

O dimensionamento correto para a criação de suportes e posicionamento dos furos nas bases possibilitou o início dos testes. A montagem final então fica como na Figura 20.

Cabe as seguintes observações:

1. Um quarto apoio foi necessário de ser adicionado devido aos testes de simulação que mostravam um desbalanço e queda do robô;

2. Cada uma das quatro barras que percorrem a altura do robô possuem em cada nível duas porcas de fixação nas partes de cima e de baixo de cada uma das bases.

3. Com a adição de todos os componentes as dimensões finais do robô tornaram-se 128mm de largura contando com as rodas e 142mm de altura.

Os suportes criados para que os sensores fiquem adequadamente posicionados são mostrados na Figura 21.

Figura 20: Vistas da modelagem do Attabot 2.0

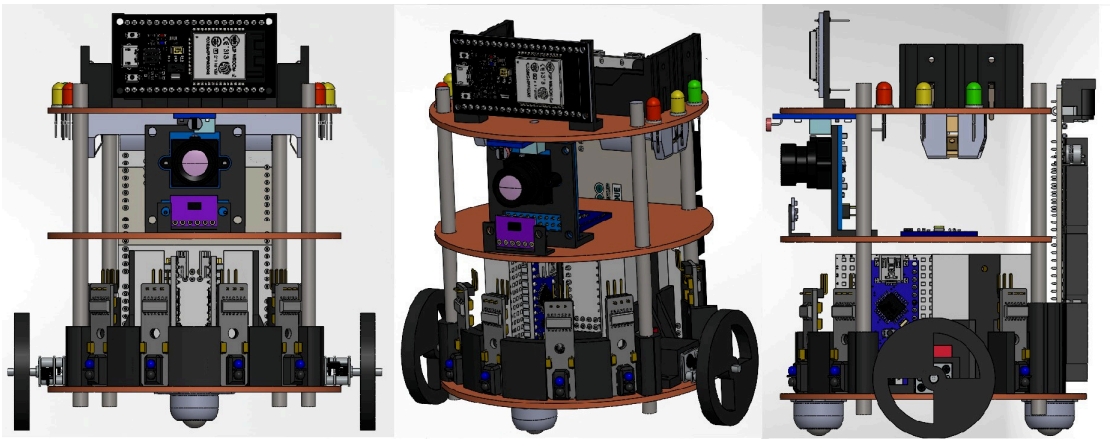
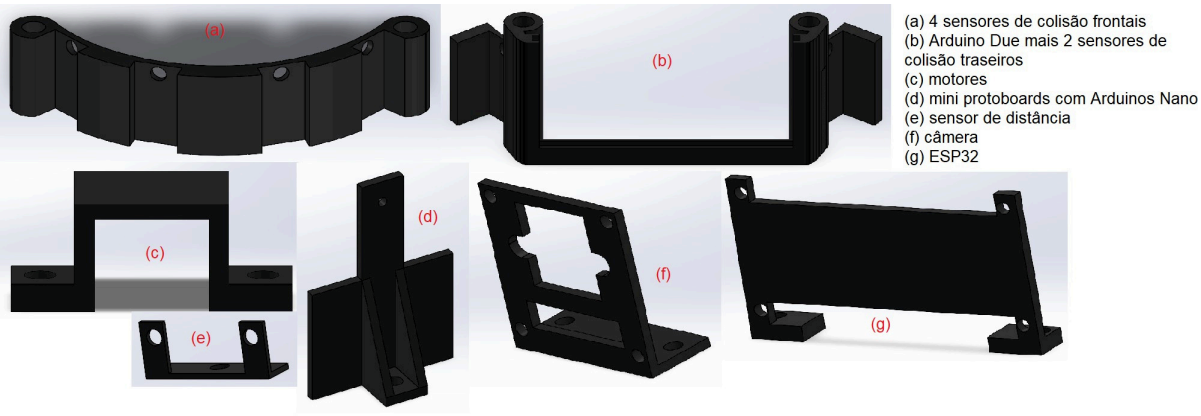


Figura 21: Suportes para os componentes



4.2 Testes

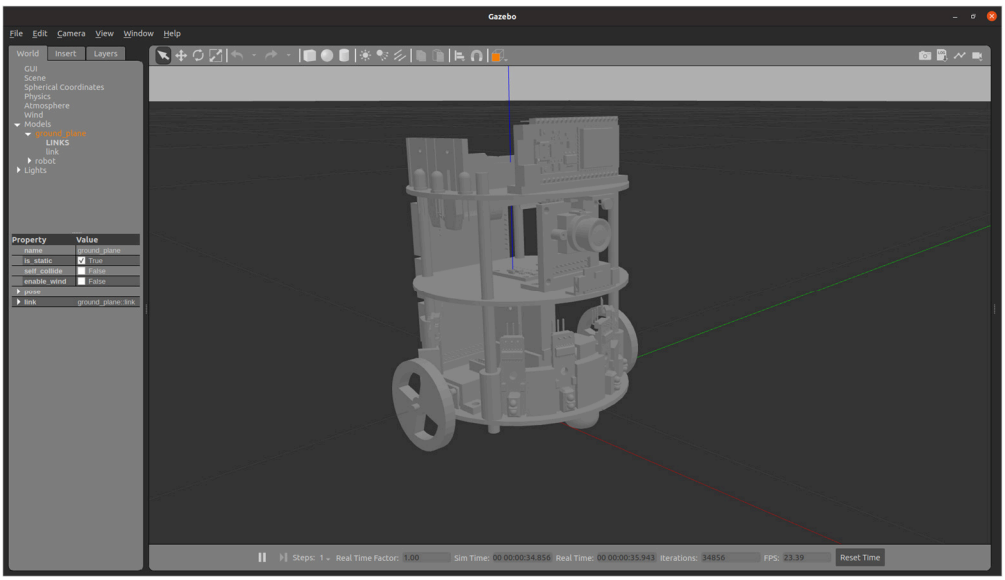
Para garantir o funcionamento desejado foi necessário a realização de alguns testes de simulação e hardware. Os testes aqui descritos são testes pontuais e não gerais.

4.2.1 Simulação

Inicialmente uma simulação pôde ser feita a partir do modelo URDF da montagem em *SolidWorks*. Utilizando o *MoveIt Setup Assistant* disponível com a instalação do *ROS* este URDF foi transformado em uma interface gráfica que permite a visualização e movimentação do robô no *Gazebo*, nesse ambiente as leis da física sao válidas e como dito em uma observação mais acima foi onde verificou-se

o desbalanço e queda do robô exigindo um quarto apoio. Cabe observar aqui que propriedades individuais de cada componente como material, massa, momento de inércia, entre outros, não foram definidas na montagem, o que nesse caso pode gerar uma diferença considerável entre simulação e modelo real.

Figura 22: Visualização em *Gazebo*



4.2.2 Teste do sensor óptico

Já para os testes de hardware o mais simples e inicialmente testado foi o TCRT5000 que possui apenas uma saída para pino digital além da alimentação e terra. O seu teste inicial consiste no acionamento de um LED e envio de uma mensagem escrita no monitor serial da IDE Arduino quando se detecta a presença de um obstáculo ao alcance do seu emissor. Com algumas modificações no código do apêndice B essa mesma mensagem é enviada ao *ROS*, o que garante nossa comunicação entre sensor e sistema.

Toda comunicaçao entre Arduino e *ROS* é inicializada em um terminal com uso do comando

```
$ rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0 _baud:=57600
```

onde _port:=/dev/ttyACM0 refere-se à porta do microcontrolador usada e _baud:=57600 à taxa de atualização da mensagem transmitida definida na IDE Arduino.

O TCRT5000 por ser de uso simples foi colocado nos testes seguintes como acionador, substituindo a função de um botão.

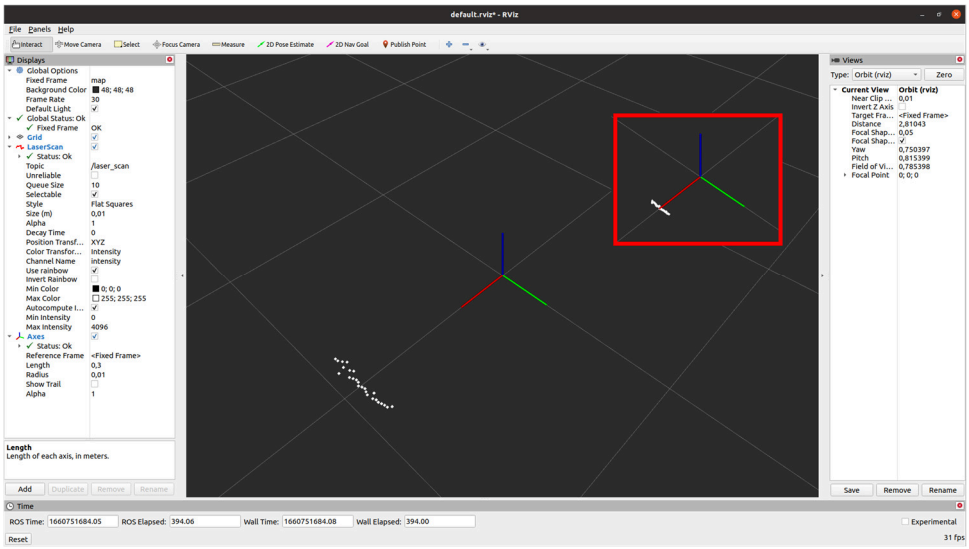
4.2.3 Comunicação I2C

O intuito deste teste foi garantir uma comunicação I2C do tipo mestre-escravo entre dispositivos onde o sensor é acionado no Arduino Due que atua como mestre e o LED é aceso no Arduino Nano que atua como escravo.

4.2.4 Teste do sensor de distância

No teste do sensor de distância laser é possível ver o alcance da medição na interface de visualização *Rviz*. A mensagem passada para o sistema é do tipo *sensor_msgs/laserScan* e exige alguns parâmetros essenciais no código como ângulo de abertura e alcance máximo e mínimo, essas informações encontram-se no da- tasheet do sensor. Na Figura 23 é possível ver a medição em duas distâncias diferentes, em ambas os 25 pontos brancos representam o vetor de posições do obstáculo detectado com o ângulo de abertura de 25º do sensor.

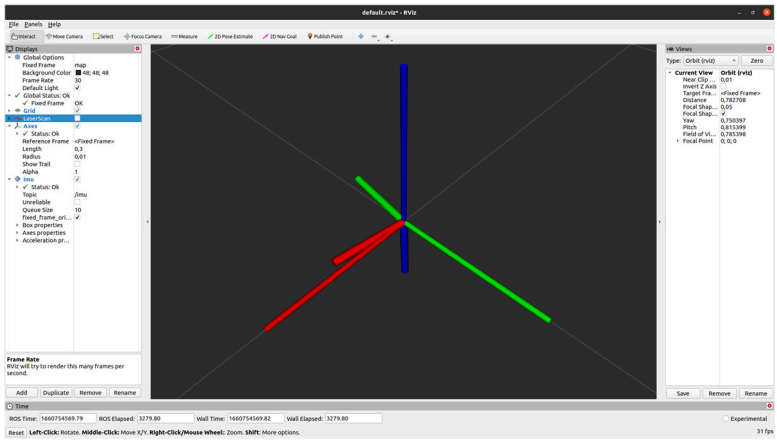
Figura 23: Visualização do sensor de distância laser em *Rviz*



4.2.5 IMU

O IMU possui mensagens do tipo *sensor_msgs/Imu*, seus parâmetros essenciais para integração em *ROS* são orientação, aceleração e velocidade e podem ser adquiridos no uso do próprio IMU dentro do código do apêndice D. Na Figura 24 o eixo maior é o mesmo eixo fixo do sensor de distância, e o eixo menor representa o IMU em uma certa posição.

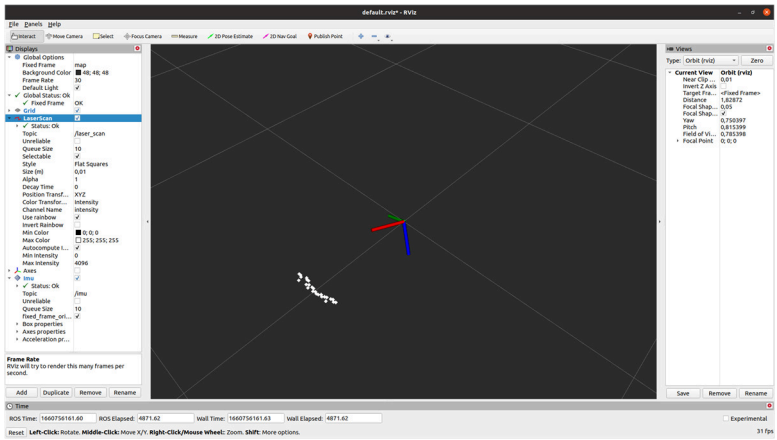
Figura 24: Visualização do IMU em *Rviz*



4.2.6 Integração VL53L0X e MPU9250

Com o IMU e o sensor de distância laser visualmente funcionando no *ROS*, houve tentativas de um funcionamento conjunto entre eles. O tempo de resposta dos dois sensores são diferentes ao ponto de um interferir no envio da mensagem do outro, a solução para este problema foi o uso da função de interrupção attachInterrupt() do Arduino. O resultado obtido se difere do resultado esperado, possivelmente devido a uma má configuração de parâmetros. Esperava-se uma alteraçao na direção do sensor de distância enquanto se movimenta o protoboard contendo ambos os sensores mas obteve-se o funcionamento individual de cada um como mostra a Figura 25.

Figura 25: Visualização do Sensor de distância e IMU em *Rviz*



4.2.7 Teste dos motores

Para os motores alguns testes distintos foram realizados, o primeiro com a intenção de fazer a comunicação mestre-escravo entre o Arduino Due mestre novamente utilizando o TCRT5000 como acionador e dois Arduinos Nano escravos contendo individualmente um motor cada.

O segundo teste objetivou o controle do giro dos motores com uso do *ROS* o que exigiu muito tempo e pesquisa para que funcionasse o mais próximo possível do necessário.

Um último teste visando a integração dos motores com o ROS possibilitou que os mesmos fossem movimentados através da ferramenta teleoperação.

4.2.8 Demais testes

Os testes com o módulo câmera, sensor de luminosidade e o ESP32 devem ser realizados futuramente e não compõem parte deste projeto. Testes mais aprofundados buscando um funcionamento mais adequado dos motores e do IMU juntamente com o sensor de distância também devem ser feitos no futuro.

5 CONCLUSÃO

Têm-se como conclusão deste projeto a real possibilidade de criação de um robô didático a partir do protótipo aqui estudado. Tendo como base projetos anteriormente desenvolvidos, os sensores aqui testados mostraram, em geral, uma boa integração com a ferramenta *ROS* dedicada exclusivamente à robótica. O baixo custo do protótipo de aproximadamente R$1050,00 e o principal software de simulação sendo de uso livre, o torna acessível para unidades de ensino que desejam investir na formação de futuros profissionais da área. O conhecimento necessário para uso da ferramenta mostrou ser de fácil aprendizagem para quem deseja, exigindo apenas um conhecimento prévio em sistemas operacionais como Linux e linguagens de programação, em especial C e C++, podendo fazer uso também de Python, porém, neste caso, o uso de Python não se aplica à programação em Arduino.

Contudo, muitas foram as dificuldades encontradas no decorrer do desenvolvimento deste projeto. Testes com os dispositivos só puderam ser realizados com a disponibilidade dos mesmos, o que acarretou em menos tempo de testes enquanto se esperava a compra e chegada dos materiais.

Por se tratar de um protótipo contendo ligações com *jumpers*, que são os fios condutores com pinos para conexão, muitos foram os problemas de mal contato que a priori não são facilmente identificados.

Nos testes com *ROS*, várias foram as vezes em que se deparava com o erro "*Unable to sync with device; possible link problem or link software version mismatch such as hydro rosserial_python with groovy Arduino*". Após diversas pesquisas com diferentes soluções, conclui-se que o erro se dá a problemas de *buffer* de memória e tamanho da mensagem passada, nenhuma das soluções foi definitiva sempre voltando a ocorrer tal erro, entretanto, desconectar o Arduino e deixá-lo descarregar totalmente solucionou o problema temporariamente durante os testes.

Outro problema encontrado foi durante o uso dos encoders que demonstrou não ser de fácil aplicação para controle da velocidade de giro dos motores o que exige uma atenção especial para que funcione exatamente como requisitado no projeto. O uso dos sensores de distância e IMU também demandaram muito tempo e pesquisa para que a visualização representada no *Rviz* se aproximasse da desejada. Devido a todo o tempo dedicado a um funcionamento adequado desses sensores e também dos motores, os testes com câmera, sensor de luminosidade e comunicação sem fio ficaram para a continuação deste projeto no futuro.

Como trabalhos futuros, além dos testes dos dispositivos acima citados e de um funcionamento mais adequado dos já funcionais, cabe reavaliar as técnicas de cálculo de velocidade pois a maioria dos sensores devem produzir a informação mais rápido que o tempo de amostragem, o encoder por outro lado tem uma particularidade de produzir a informação inversamente proporcional à velocidade de rotaçã. Cabe também o estudo do *ROS 2* que por ser recente ainda se

encontra em desenvolvimento, e portanto não possui uma comunidade tão grande quanto sua primeira versão, entretanto, para aplicações que envolvam diversos robôs se comunicando entre si, comportamento esse conhecido como enxame de robôs, o *ROS 2* possui as ferramentas adequadas.

Por fim, um melhor estudo sobre as baterias afim de identificar a necessidade ou não de baterias com maiores cargas para testes de bancada com maior tempo de autonomia.

REFERÊNCIAS

O que é robótica e como ela está presente no seu dia-a-dia? **Portal da indústria**. Disponível em: <https://www.portaldaindustria.com.br/industria-de-a-z/robotica/#:~:text=Rob%C3%B3tica%20%C3%A9%20a%20ci%C3%AAncia%20que,movimentos%20humanos%20simples%20ou%20complexos.>

Pio, J. L. S.; Castro, T. H. C. ; Castro Junior, A. N. **A Robótica Móvel como Instrumento de Aprendizagem em Ciência da Computação**. In: XVII Simpósio Brasileiro de Informática na Educação, 2006, Rio de Janeiro. XVII SBIE. Campinas/SP: SBC. v. 1. p. 197-206.

Simões, A. S. ; Carrion, R. ; Martins, A C G ; Franchin, M N . Utilizando a plataforma LEGO Mindstorm em disciplinas do ciclo básico do curso de Engenharia Mecatrônica. In: XXVI Congresso da Sociedade Brasileira de Computação (SBC), 2006, Campo Grande. ENRI'06: Encontro de Robótica Inteligente, 2006.

O que é uma IDE? **Alura. Disponível em:** <https://www.alura.com.br/artigos/o-que-e-uma-ide>

Arduino Due. Disponível em: <https://docs.arduino.cc/hardware/due>

Arduino Nano. Disponível em: <https://docs.arduino.cc/hardware/nano>

Interface gráfica para comunicação serial UART. **Embarcados**. Disponível em: <https://embarcados.com.br/interface-grafica-para-uart/>

I2C. **Mundo projetado**. Disponível em <https://mundoprojetado.com.br/i2c/>

SPI. **Mundo projetado.** Disponível em: <https://mundoprojetado.com.br/spi/>

O que é protocolo MQTT? **HI Tecnologia.** Disponível em: <https://materiais.hitecnologia.com.br/blog/o-que-e-protocolo-mqtt/>

Conhecendo o NodeMCU-32S ESP32. **MasterWalker Eletronic Shop.** Disponível em: <https://blogmasterwalkershop.com.br/embarcados/esp32/conhecendo-o-nodemcu-32s-esp32>

Visão geral do encoder de quadratura. **Dynapar encoders.** Disponível em: <https://www.dynaparencoders.com.br/blog/visao-geral-do-encoder-de-quadratura/>

Visão geral sobre encoders de quadratura. **Mokka-sensors.** Disponível em: < https://www.mokka-sensors.com.br/2020/03/27/encoders-de-quadratura-mokka/>

Ponte H – O que é e como funciona. **Mundo projetado.** Disponível em: <https://mundoprojetado.com.br/ponte-h-o-que-e-e-como-funciona/>

O que é PWM e para que serve. **Citisystems.** Disponível em: <https://www.citisystems.com.br/pwm/>

Adafruit DRV8871 Brushed DC Motor Driver Breakout. **Learn Adafruit.** Disponível em:

<https://learn.adafruit.com/adafruit-drv8871-brushed-dc-motor-driver-breakout>

Módulo seguidor de linha / Faixa infravermelho TCRT5000. **Casa da robótica.** Disponível em: <https://www.casadarobotica.com/sensores-modulos/modulos/outros/modulo-seguidor-de-linha-faixa-infravermelho-tcrt5000-1-canal>

Módulo acelerômetro e giroscópio 9 eixos mpu-9250 . **Saravati.** Disponível em: <https://www.saravati.com.br/modulo-acelerometro-e-giroscopio-9-eixos-mpu-9250-gy-9250>

Sensor de distância VL53L0X de alta precisão. **Baú da eletrônica.** Disponível em: <https://www.baudaeletronica.com.br/sensor-de-distancia-vl53l0x-de-alta-precis-o.html>

Módulo câmera VGA OV7670. **Filipeflop.** Disponível em: <https://www.filipeflop.com/blog/modulo-camera-vga-ov7670/>

Módulo Sensor de luminosidade LDR. **Eletrogate.** Disponível em: < https://www.eletrogate.com/modulo-sensor-de-luminosidade-ldr>

Bateria 18650 Li-Ion recarregável. **Usinainfo.** Disponível em: <https://www.usinainfo.com.br/baterias/bateria-18650-li-ion-recarregavel-37v-3800mah-button-top-5072.html>

SolidWorks. Disponível em: <https://www.solidworks.com/pt-br>

Ros documentation. Disponível em: <http://wiki.ros.org/Documentation>

What is a ROS Node? **The Robotics Back-End.** Disponível em: <https://roboticsbackend.com/what-is-a-ros-node/>

What is the difference between Rviz and Gazebo? **Automatic Addison.** Disponível em: <https://automaticaddison.com/what-is-the-difference-between-rviz-and-gazebo/>

SolidWorks to URDF exporter. Disponível em: <http://wiki.ros.org/sw_urdf_exporter>

Lages, W. F**. Unified Robot Description Format (URDF)**. Universidade Federal do Rio Grande do Sul. Disponível em: <http://www.ece.ufrgs.br/~fetter/eng10026/urdf.pdf>

Ferreira, M. V. M. **Aplicação da iPNRD com feromônio de formiga para controle e distrubuído de robôs móveis.** Universidade Federal de Uberlândia, 2020. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/32643/1/Aplica%c3%a7%c3%a3oiPNRDFerom%c3%b4nio.pdf>

e-puck education robot. Disponível em: <https://e-puck.gctronic.com/>

How to publish wheel encoder tick data using ROS and Arduino. **Automatic Addison**. Disponível em: <https://automaticaddison.com/how-to-publish-wheel-encoder-tick-data-using-ros-and-arduino/>

APÊNDICE

APÊNDICE A - Arquivos CAD dos suportes

Os suportes criados para fixar os componentes do robô encontram-se no link https://github.com/MAPL-UFU/Attabot-2.0/tree/main/CAD/suportes

APÊNDICE B - Código do Sensor óptico

```
//sempre que houver um evento de quase colisao envia um sinal

#define USE_USBCON
#include <ros.h>
#include <std_msgs/String.h>

const int pin_SENSOR = 12;

const int pin_LED =  13;

int sensorState = 0;

int prestate = 0;

ros::NodeHandle  nh;

std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);

char event[13] = "EVENT DETECT";

void setup() {

  pinMode(pin_LED, OUTPUT);

  pinMode(pin_SENSOR, INPUT);

  nh.getHardware()->setBaud(9600);
  nh.initNode();
  nh.advertise(chatter);

}

void loop() {

  sensorState = digitalRead(pin_SENSOR);

  if (sensorState == HIGH && prestate == 0) {
```

```
39
        digitalWrite(pin_LED, HIGH);
41
        prestate = 1;
43
        str_msg.data = event;
45      chatter.publish( &str_msg );
        nh.spinOnce();
47
        delay(500);
49
    } else if (sensorState == LOW) {
51
        digitalWrite(pin_LED, LOW);
53
        prestate = 0;
55
    }
57
}
```

tcrt5000.ino

APÊNDICE C – Código do Sensor de distância

```
#include <Wire.h>
2 #include <VL53L0X.h>

4 #define USE_USBCON
  #include <ros.h>
6 #include <sensor_msgs/LaserScan.h>

8 //#define HIGH_SPEED
  //#define HIGH_ACCURACY
10
  // Cria uma instancia do sensor
12 VL53L0X sensor;

14 ros::NodeHandle  nh;

16 sensor_msgs::LaserScan lidar_msg;
  ros::Publisher lidar_pub("/laser_scan", &lidar_msg);
18
  float ranges[25] = {0};
20 int p = 0;
  float angle_min;
22 float angle_max;
  const float Pi = 3.14159;
```

```
bool take_measure = true;
//bool begin_measure = false;

void setup()
{
  // Inicializa a comunicacao serial
  //Serial.begin(9600);
  // Inicializa a comunicacao I2C
  Wire.begin();

  #if defined HIGH_SPEED
  // reduce timing budget to 20 ms (default is about 33 ms)
  sensor.setMeasurementTimingBudget(20000);
  #elif defined HIGH_ACCURACY
  // increase timing budget to 200 ms
  sensor.setMeasurementTimingBudget(200000);
  #endif

  nh.getHardware()->setBaud(57600);
  nh.initNode();
  nh.advertise(lidar_pub);

  // Inicializa o sensor
  sensor.init();
//  sensor.startContinuous();
  // Define um timeout de 500mS para a leitura do sensor
  // Em caso de erro, este sera o tempo maximo de espera da
   resposta do sensor
  sensor.setTimeout(500);
}

void loop(){
//  while(begin_measure == true) {
  p = 0;

  lidar_msg.header.frame_id = "laser";
  lidar_msg.angle_min = -(12.5*Pi)/180;
  lidar_msg.angle_max = (12.5*Pi)/180;
  lidar_msg.angle_increment = (1*Pi)/180;
  lidar_msg.range_min = 0.05;          // minimum range value [m]
  lidar_msg.range_max = 1.2;           // maximum range value [m]

  unsigned long scan_start = millis();

  while(p <= 24) {
    if(take_measure == true) {
    ranges[p] = (float)sensor.readRangeSingleMillimeters()/1000.0;
    ++p;
    }
```

```
  }

  lidar_msg.scan_time = (millis()-scan_start)/1000.0;
//  lidar_msg.time_increment = 0.5;
  lidar_msg.ranges_length = p;
  lidar_msg.ranges = ranges;
  lidar_msg.header.stamp = nh.now();
  lidar_pub.publish(&lidar_msg);
  nh.advertise(lidar_pub);
  nh.spinOnce();
//  }
}

//void control_measure(){
//    if(begin_measure == false) {
//      begin_measure = true;
//    }
//    else  {
//      begin_measure = false;
//    }
//}
```

vl53l0x.ino

APÊNDICE D – Código do IMU

```
/*
biblioteca: https://github.com/hideakitai/MPU9250
doc: http://docs.ros.org/en/lunar/api/sensor_msgs/html/msg/Imu.
   html
datasheet: https://invensense.tdk.com/wp-content/uploads/2015/02/
   PS-MPU-9250A-01-v1.1.pdf
example: https://github.com/dheera/ros-imu-bno055/blob/master/src/
   bno055_i2c_node.cpp
requer: sudo apt-get install ros-noetic-imu-tools
*/

#include "MPU9250.h"

#define USE_USBCON
#include <ros.h>
#include <sensor_msgs/Imu.h>

MPU9250 mpu;

ros::NodeHandle nh;

sensor_msgs::Imu imu_msg;
```

```
ros::Publisher imu_pub("imu", &imu_msg);


void setup() {
    Serial.begin(115200);
    Wire.begin();
    delay(2000);

    if (!mpu.setup(0x68)) {  // change to your own address
        while (1) {
            Serial.println("MPU connection failed. Please check your connection with 'connection_check' example.");
            delay(5000);
        }
    }

    nh.getHardware()->setBaud(57600);
    nh.initNode();
    nh.advertise(imu_pub);
}

void loop() {

    nh.spinOnce();
    if (mpu.update()) {
        static uint32_t prev_ms = millis();
        if (millis() > prev_ms + 25) {
            print_roll_pitch_yaw();
            prev_ms = millis();
        }
    }
    imu_msg.header.frame_id = "imu";
    imu_msg.header.stamp = nh.now();

    imu_msg.orientation.x = mpu.getEulerX()-178;
    imu_msg.orientation.y = mpu.getEulerY();
    imu_msg.orientation.z = mpu.getEulerZ()-43;
    imu_msg.orientation.w = mpu.getQuaternionW();

    imu_msg.linear_acceleration.x = mpu.getLinearAccX();
    imu_msg.linear_acceleration.y = mpu.getLinearAccY();
    imu_msg.linear_acceleration.z = mpu.getLinearAccZ();

    imu_msg.angular_velocity.x = mpu.getGyroX();
    imu_msg.angular_velocity.y = mpu.getGyroY();
    imu_msg.angular_velocity.z = mpu.getGyroZ();

    imu_pub.publish(&imu_msg);
    nh.advertise(imu_pub);
```

```
68 }

70 void print_roll_pitch_yaw() {
       Serial.print("Yaw, Pitch, Roll: ");
72     Serial.print(mpu.getYaw(), 0);
       Serial.print(", ");
74     Serial.print(mpu.getPitch(), 0);
       Serial.print(", ");
76     Serial.println(mpu.getRoll(), 0);
   }
```

mpu9250.ino

APÊNDICE E – Integração entre sensor de distância e IMU

```
1  #include "MPU9250.h"
   #include <VL53LOX.h>
3
   #define USE_USBCON
5  #include <ros.h>
   #include <sensor_msgs/Imu.h>
7  #include <sensor_msgs/LaserScan.h>

9  int intPin=4;
   int intPin2=5;   // interrupt pin
11 MPU9250 mpu;
   VL53LOX sensor;
13
   ros::NodeHandle nh;
15
   sensor_msgs::LaserScan lidar_msg;
17 ros::Publisher lidar_pub("/laser_scan", &lidar_msg);

19 sensor_msgs::Imu imu_msg;
   ros::Publisher imu_pub("imu", &imu_msg);
21


23 float ranges[25] = {0};
   volatile int p = 0;
25 float angle_min;
   float angle_max;
27 const float Pi = 3.14159;
   bool take_measure = true;
29


31 void setup() {
     Wire.begin();
33   delay(2000);
```

```
  if (!mpu.setup(0x68)) {  // change to your own address
      while (1) {
          Serial.println("MPU connection failed. Please check your connection with 'connection_check' example.");
          delay(5000);
      }
  }
  attachInterrupt(digitalPinToInterrupt(intPin), getImu, RISING);
  //attachInterrupt(digitalPinToInterrupt(intPin2), getScanSensor, RISING);

  nh.getHardware()->setBaud(57600);
  nh.initNode();
  nh.advertise(imu_pub);
  nh.advertise(lidar_pub);

  sensor.init();
  sensor.setTimeout(500);
}

void loop() {

  getScanSensor();
  //getImu();

  lidar_pub.publish(&lidar_msg);
  nh.advertise(lidar_pub);
  imu_pub.publish(&imu_msg);
  nh.advertise(imu_pub);
  nh.spinOnce();
}

void getImu(){

  if (mpu.update()) {
      volatile static uint32_t prev_ms = millis();
      if (millis() > prev_ms + 25) {
          prev_ms = millis();
      }
  }

  imu_msg.header.frame_id = "imu";


  imu_msg.orientation.x = mpu.getQuaternionX();
  imu_msg.orientation.y = mpu.getQuaternionY();
  imu_msg.orientation.z = mpu.getQuaternionZ();
  imu_msg.orientation.w = mpu.getQuaternionW();
```

```
81
   imu_msg.linear_acceleration.x = mpu.getLinearAccX();
83 imu_msg.linear_acceleration.y = mpu.getLinearAccY();
   imu_msg.linear_acceleration.z = mpu.getLinearAccZ();
85 //imu_msg.linear_acceleration_covariance[0] = -1;

87 imu_msg.angular_velocity.x = mpu.getRoll();
   imu_msg.angular_velocity.y = mpu.getPitch();
89 imu_msg.angular_velocity.z = mpu.getYaw();

91 imu_msg.header.stamp = nh.now();

93 }

95 void getScanSensor(){

97 int p = 0;

99 lidar_msg.header.frame_id = "laser";
   lidar_msg.angle_min = -(12.5*Pi)/180;
101 lidar_msg.angle_max = (12.5*Pi)/180;
   lidar_msg.angle_increment = (1*Pi)/180;
103 lidar_msg.range_min = 0.05;          // minimum range value [m]
   lidar_msg.range_max = 1.2;           // maximum range value [m]
105
   volatile unsigned long scan_start = millis();
107
   while(p <= 24) {
109  if(take_measure == true) {
     ranges[p] = (volatile float)sensor.readRangeSingleMillimeters
    ()/1000.0;
111  ++p;
     }
113 }

115 lidar_msg.scan_time = (millis()-scan_start)/1000.0;
//  lidar_msg.time_increment = 0.5;
117 lidar_msg.ranges_length = p;
   lidar_msg.ranges = ranges;
119 lidar_msg.header.stamp = nh.now();

121
}
```

laserScan_Imu.ino

APÊNDICE F - Código do motor Mestre

```
//Codigo I2C Mestre para o arduino due

#include "Wire.h"

const int pin_SENSOR = 12;

int sensorState = 0;
int prestate = 0;

#define slaveAdress 0x08


void setup() {
  Wire.begin();
  pinMode(pin_SENSOR, INPUT);

}

void loop() {
  sensorState = digitalRead(pin_SENSOR);

  if (sensorState == HIGH && prestate == 0) {

    Wire.beginTransmission(slaveAdress);
    Wire.write(sensorState);
    Wire.endTransmission();

    prestate = 1;

    delay(500);

  } else if (sensorState == LOW) {

    Wire.beginTransmission(slaveAdress);
    Wire.write(sensorState);
    Wire.endTransmission();

    prestate = 0;

  }

}
```

i2c_motor_master.ino

APÊNDICE G – Código do motor Escravo

```
//Codigo I2C Escravo para o arduino nano

#include "Wire.h"


#define mL 8
#define mR 9
#define pin_LED 3
#define myAdress 0x08

void setup() {
  Wire.begin(myAdress);
  Wire.onReceive(receiveEvent);

  pinMode(mL, OUTPUT);
  pinMode(mR, OUTPUT);
  pinMode(pin_LED, OUTPUT);

  Serial.begin(9600);
}

void loop() {


}

void receiveEvent(int howMany) {
  // verifica se existem dados para serem lidos no barramento I2C
  if (Wire.available()) {
    // le o byte recebido
    char received = Wire.read();

    // se o byte recebido for igual a 0, apaga o LED
    if (received == 0) {
     digitalWrite(mL, LOW);
     digitalWrite(mR, LOW);
     digitalWrite(pin_LED, LOW);
    }

    // se o byte recebido for igual a 1 acende o LED
    if (received == 1) {
      digitalWrite(mL, LOW);
      digitalWrite(mR, HIGH);
      digitalWrite(pin_LED, HIGH);
    }
    Serial.println(received);
  }
}
```

i2c_motor_slave.ino

APÊNDICE H – Integração dos motores com ROS

```
/*
 * Author: Automatic Addison
 * Website: https://automaticaddison.com
 * Description: ROS node that publishes the accumulated ticks for
   each wheel
 * (/right_ticks and /left_ticks topics) at regular intervals
   using the
 * built-in encoder (forward = positive; reverse = negative).
 * The node also subscribes to linear & angular velocity commands
   published on
 * the /cmd_vel topic to drive the robot accordingly.
 * Reference: Practical Robotics in C++ book (ISBN-10 :
   9389423465)
 */

#include <ros.h>
#include <std_msgs/Int16.h>
#include <geometry_msgs/Twist.h>

// Handles startup and shutdown of ROS
ros::NodeHandle nh;

////////////////// Tick Data Publishing Variables and Constants
   ///////////////

// Encoder output to Arduino Interrupt pin. Tracks the tick count.
#define ENC_IN_LEFT_A 2
#define ENC_IN_RIGHT_A 3

// Other encoder output to Arduino to keep track of wheel
   direction
// Tracks the direction of rotation.
#define ENC_IN_LEFT_B 4
#define ENC_IN_RIGHT_B 11

// True = Forward; False = Reverse
boolean Direction_left = true;
boolean Direction_right = true;

// Minumum and maximum values for 16-bit integers
// Range of 65,535
const int encoder_minimum = -32768;
```

```
const int encoder_maximum = 32767;

// Keep track of the number of wheel ticks
std_msgs::Int16 right_wheel_tick_count;
ros::Publisher rightPub("right_ticks", &right_wheel_tick_count);

std_msgs::Int16 left_wheel_tick_count;
ros::Publisher leftPub("left_ticks", &left_wheel_tick_count);

// Time interval for measurements in milliseconds
const int interval = 30;
long previousMillis = 0;
long currentMillis = 0;

////////////////// Motor Controller Variables and Constants ///////////////////

// Motor A connections
const int enA = 9;
const int in1 = 5;
const int in2 = 6;

// Motor B connections
const int enB = 10;
const int in3 = 7;
const int in4 = 8;

// How much the PWM value can change each cycle
const int PWM_INCREMENT = 1;

// Number of ticks per wheel revolution. We won't use this in this code.
const int TICKS_PER_REVOLUTION = 620;

// Wheel radius in meters
const double WHEEL_RADIUS = 0.033;

// Distance from center of the left tire to the center of the right tire in m
const double WHEEL_BASE = 0.17;

// Number of ticks a wheel makes moving a linear distance of 1 meter
// This value was measured manually.
const double TICKS_PER_METER = 3100; // Originally 2880

// Proportional constant, which was measured by measuring the
// PWM-Linear Velocity relationship for the robot.
const int K_P = 278;
```

```
82
// Y-intercept for the PWM-Linear Velocity relationship for the
   robot
84 const int b = 52;

86 // Correction multiplier for drift. Chosen through experimentation
   .
   const int DRIFT_MULTIPLIER = 120;
88
   // Turning PWM output (0 = min, 255 = max for PWM values)
90 const int PWM_TURN = 80;

92 // Set maximum and minimum limits for the PWM values
   const int PWM_MIN = 80; // about 0.1 m/s
94 const int PWM_MAX = 100; // about 0.172 m/s

96 // Set  linear velocity and PWM variable values for each wheel
   double velLeftWheel = 0;
98 double velRightWheel = 0;
   double pwmLeftReq  = 0;
100 double pwmRightReq = 0;

102 // Record the time that the last velocity command was received
   double lastCmdVelReceived = 0;
104
   ////////////////////// Tick Data Publishing Functions
      ///////////////////////
106
   // Increment the number of ticks
108 void right_wheel_tick() {

110   // Read the value for the encoder for the right wheel
      int val = digitalRead(ENC_IN_RIGHT_B);
112
      if (val == LOW) {
114     Direction_right = false; // Reverse
      }
116   else {
        Direction_right = true; // Forward
118   }

120   if (Direction_right) {

122     if (right_wheel_tick_count.data == encoder_maximum) {
          right_wheel_tick_count.data = encoder_minimum;
124     }
        else {
126       right_wheel_tick_count.data++;
        }
```

```
128   }
      else {
130     if (right_wheel_tick_count.data == encoder_minimum) {
          right_wheel_tick_count.data = encoder_maximum;
132     }
        else {
134       right_wheel_tick_count.data--;
        }
136   }
}
138
// Increment the number of ticks
140 void left_wheel_tick() {

142   // Read the value for the encoder for the left wheel
      int val = digitalRead(ENC_IN_LEFT_B);
144
      if (val == LOW) {
146     Direction_left = true; // Reverse
      }
148   else {
        Direction_left = false; // Forward
150   }

152   if (Direction_left) {
        if (left_wheel_tick_count.data == encoder_maximum) {
154       left_wheel_tick_count.data = encoder_minimum;
        }
156     else {
          left_wheel_tick_count.data++;
158     }
      }
160   else {
        if (left_wheel_tick_count.data == encoder_minimum) {
162       left_wheel_tick_count.data = encoder_maximum;
        }
164     else {
          left_wheel_tick_count.data--;
166     }
      }
168 }

170 ////////////////////// Motor Controller Functions
       ///////////////////////////

172 // Calculate the left wheel linear velocity in m/s every time a
    // tick count message is rpublished on the /left_ticks topic.
174 void calc_vel_left_wheel(){
```

```
176   // Previous timestamp
      static double prevTime = 0;
178
      // Variable gets created and initialized the first time a
       function is called.
180   static int prevLeftCount = 0;

182   // Manage rollover and rollunder when we get outside the 16-bit
       integer range
      int numOfTicks = (65535 + left_wheel_tick_count.data -
       prevLeftCount) %65535;
184
      // If we have had a big jump, it means the tick count has rolled
        over.
186   if (numOfTicks > 10000) {
            numOfTicks = 0 - (65535 - numOfTicks);
188   }

190   // Calculate wheel velocity in meters per second
      velLeftWheel = numOfTicks/TICKS_PER_METER/((millis()/1000)-
       prevTime);
192
      // Keep track of the previous tick count
194   prevLeftCount = left_wheel_tick_count.data;

196   // Update the timestamp
      prevTime = (millis()/1000);
198
}
200
// Calculate the right wheel linear velocity in m/s every time a
202 // tick count message is published on the /right_ticks topic.
void calc_vel_right_wheel(){
204
      // Previous timestamp
206   static double prevTime = 0;

208   // Variable gets created and initialized the first time a
       function is called.
      static int prevRightCount = 0;
210
      // Manage rollover and rollunder when we get outside the 16-bit
       integer range
212   int numOfTicks = (65535 + right_wheel_tick_count.data -
       prevRightCount) % 65535;

214   if (numOfTicks > 10000) {
            numOfTicks = 0 - (65535 - numOfTicks);
216   }
```

```
218   // Calculate wheel velocity in meters per second
      velRightWheel = numOfTicks/TICKS_PER_METER/((millis()/1000)-
       prevTime);
220
      prevRightCount = right_wheel_tick_count.data;
222
      prevTime = (millis()/1000);
224
    }
226
    // Take the velocity command as input and calculate the PWM values
       .
228 void calc_pwm_values(const geometry_msgs::Twist& cmdVel) {

230   // Record timestamp of last velocity command received
      lastCmdVelReceived = (millis()/1000);
232
      // Calculate the PWM value given the desired velocity
234   pwmLeftReq = K_P * cmdVel.linear.x + b;
      pwmRightReq = K_P * cmdVel.linear.x + b;
236
      // Check if we need to turn
238   if (cmdVel.angular.z != 0.0) {

240     // Turn left
        if (cmdVel.angular.z > 0.0) {
242       pwmLeftReq =-PWM_TURN;
          pwmRightReq = PWM_TURN;
244     }
        // Turn right
246     else {
          pwmLeftReq  = PWM_TURN;
248       pwmRightReq = -PWM_TURN;
        }
250   }
      // Go straight
252   else {

254     // Remove any differences in wheel velocities
        // to make sure the robot goes straight
256     static double prevDiff = 0;
        static double prevPrevDiff = 0;
258     double currDifference = velLeftWheel - velRightWheel;
        double avgDifference = (prevDiff+prevPrevDiff+currDifference)
       /3;
260     prevPrevDiff = prevDiff;
        prevDiff = currDifference;
262
```

```
    // Correct PWM values of both wheels to make the vehicle go
    straight
    pwmLeftReq -= (int)(avgDifference * DRIFT_MULTIPLIER);
    pwmRightReq += (int)(avgDifference * DRIFT_MULTIPLIER);
  }

  // Handle low PWM values
  if (abs(pwmLeftReq) < PWM_MIN) {
    pwmLeftReq = 0;
  }
  if (abs(pwmRightReq) < PWM_MIN) {
    pwmRightReq = 0;
  }
}

void set_pwm_values() {

  // These variables will hold our desired PWM values
  static int pwmLeftOut = 0;
  static int pwmRightOut = 0;

  // If the required PWM is of opposite sign as the output PWM, we
    want to
  // stop the car before switching direction
  static bool stopped = false;
  if ((pwmLeftReq * velLeftWheel < 0 && pwmLeftOut != 0) ||
      (pwmRightReq * velRightWheel < 0 && pwmRightOut != 0)) {
    pwmLeftReq = 0;
    pwmRightReq = 0;
  }

  // Set the direction of the motors
  if (pwmLeftReq > 0) { // Left wheel forward
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
  }
  else if (pwmLeftReq < 0) { // Left wheel reverse
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
  }
  else if (pwmLeftReq == 0 && pwmLeftOut == 0 ) { // Left wheel
    stop
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
  }
  else { // Left wheel stop
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
  }
```

```
  if (pwmRightReq > 0) { // Right wheel forward
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  }
  else if(pwmRightReq < 0) { // Right wheel reverse
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
  }
  else if (pwmRightReq == 0 && pwmRightOut == 0) { // Right wheel
   stop
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
  }
  else { // Right wheel stop
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
  }

  // Increase the required PWM if the robot is not moving
  if (pwmLeftReq != 0 && velLeftWheel == 0) {
    pwmLeftReq *= 1.5;
  }
  if (pwmRightReq != 0 && velRightWheel == 0) {
    pwmRightReq *= 1.5;
  }

  // Calculate the output PWM value by making slow changes to the
   current value
  if (abs(pwmLeftReq) > pwmLeftOut) {
    pwmLeftOut += PWM_INCREMENT;
  }
  else if (abs(pwmLeftReq) < pwmLeftOut) {
    pwmLeftOut -= PWM_INCREMENT;
  }
  else{}

  if (abs(pwmRightReq) > pwmRightOut) {
    pwmRightOut += PWM_INCREMENT;
  }
  else if(abs(pwmRightReq) < pwmRightOut) {
    pwmRightOut -= PWM_INCREMENT;
  }
  else{}

  // Conditional operator to limit PWM output at the maximum
  pwmLeftOut = (pwmLeftOut > PWM_MAX) ? PWM_MAX : pwmLeftOut;
  pwmRightOut = (pwmRightOut > PWM_MAX) ? PWM_MAX : pwmRightOut;
```

```
  // PWM output cannot be less than 0
  pwmLeftOut = (pwmLeftOut < 0) ? 0 : pwmLeftOut;
  pwmRightOut = (pwmRightOut < 0) ? 0 : pwmRightOut;

  // Set the PWM value on the pins
  analogWrite(enA, pwmLeftOut);
  analogWrite(enB, pwmRightOut);
}

// Set up ROS subscriber to the velocity command
ros::Subscriber<geometry_msgs::Twist> subCmdVel("cmd_vel", &
    calc_pwm_values );

void setup() {

  // Set pin states of the encoder
  pinMode(ENC_IN_LEFT_A , INPUT_PULLUP);
  pinMode(ENC_IN_LEFT_B , INPUT);
  pinMode(ENC_IN_RIGHT_A , INPUT_PULLUP);
  pinMode(ENC_IN_RIGHT_B , INPUT);

  // Every time the pin goes high, this is a tick
  attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A),
    left_wheel_tick, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A),
    right_wheel_tick, RISING);

  // Motor control pins are outputs
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  // Turn off motors - Initial state
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);

  // Set the motor speed
  analogWrite(enA, 0);
  analogWrite(enB, 0);

  // ROS Setup
  nh.getHardware()->setBaud(115200);
  nh.initNode();
  nh.advertise(rightPub);
```

```
402   nh.advertise(leftPub);
      nh.subscribe(subCmdVel);
404 }

406 void loop() {

408   nh.spinOnce();

410   // Record the time
      currentMillis = millis();
412
      // If the time interval has passed, publish the number of ticks,
414   // and calculate the velocities.
      if (currentMillis - previousMillis > interval) {
416
        previousMillis = currentMillis;
418
        // Publish tick counts to topics
420     leftPub.publish( &left_wheel_tick_count );
        rightPub.publish( &right_wheel_tick_count );
422
        // Calculate the velocity of the right and left wheels
424     calc_vel_right_wheel();
        calc_vel_left_wheel();
426
      }
428
      // Stop the car if there are no cmd_vel messages
430   if((millis()/1000) - lastCmdVelReceived > 1) {
        pwmLeftReq  = 0;
432     pwmRightReq = 0;
      }
434
      set_pwm_values();
436 }
```

motor_ROS.ino