CENTRO UNIVERSITÁRIO FEI

DIMAS LOURENÇO DA SILVA JUNIOR

FABIO REINA DA SILVA

RENATO SILVA DE LIMA

RODRIGO BENICIO COELHO

CADEIRA DE RODAS AUTÔNOMA COMANDADA POR APLICATIVO

São Bernardo do Campo

2021

DIMAS LOURENÇO DA SILVA JUNIOR
FABIO REINA DA SILVA
RENATO SILVA DE LIMA
RODRIGO BENICIO COELHO

CADEIRA DE RODAS AUTÔNOMA COMANDADA POR APLICATIVO

Trabalho de Conclusão de Curso apresentado ao Centro Universitário FEI, como parte dos requisitos necessários para obtenção do título de Bacharel em Engenharia Elétrica. Orientado pelo Prof. Dr. Isaac Jesus da Silva.

São Bernardo do Campo

2021

Cadeira de rodas autônoma comandada por aplicativo / Dimas Lourenço Da Silva Junior...[et al.]. São Bernardo do Campo, 2021.
60 f. : il.

Trabalho de Conclusão de Curso - Centro Universitário FEI.
Orientador: Prof. Dr. Isaac Jesus da Silva.

1. Acessibilidade. 2. Navegação autônoma. 3. Cadeira de rodas. 4. ROS. I. Lourenço Da Silva Junior, Dimas. II. Reina Da Silva, Fábio. III. Silva De Lima, Renato. IV. Benicio Coelho, Rodrigo. V. Jesus da Silva, Isaac, orient. VI. Título.

Elaborada pelo sistema de geração automática de ficha catalográfica da FEI com os dados fornecidos pelo(a) autor(a).

DIMAS LOURENÇO DA SILVA JUNIOR
FABIO REINA DA SILVA
RENATO SILVA DE LIMA
RODRIGO BENICIO COELHO

CADEIRA DE RODAS AUTÔNOMA COMANDADA POR APLICATIVO

Trabalho de Conclusão de Curso, apresentado ao Centro Universitário FEI, como parte dos requisitos necessários para obtenção do título de Bacharel em Engenharia Elétrica.

Comissão julgadora

_____
Orientador

_____
Examinador (1)

_____
Examinador (2)

São Bernardo do Campo

2021

RESUMO

As políticas de acessibilidade no Brasil estão longe de serem ideais. Quase 7% da população brasileira apresenta uma deficiência locomotora segundo censo de 2010 do IBGE. Analisando o cenário descrito, o objetivo do projeto é melhorar uma cadeira de rodas motorizada, incluindo funcionalidades para que a maior parte desse público seja abrangida e possa desfrutar de um transporte que exige menos esforços. No software, o projeto utilizou o ROS, que fornece um conjunto de frameworks para desenvolvimento de robôs, e um aplicativo Android como interface do sistema. Já no hardware, como não será montado um protótipo físico, o simulador Gazebo foi utilizado para apresentar o comportamento do mesmo em conjunto com o programa. A modelagem do hardware também foi descrita em caso de uma montagem física futura, e é composta por motores, microprocessador Raspberry, Arduino, reguladores de tensão e LIDAR. O projeto por tanto propõe usar diferentes tecnologias acessíveis para criar uma cadeira de rodas motorizada que permita ao usuário ter uma melhor mobilidade, ao utilizar um aplicativo de celular que o possibilitará usar duas opções de controle diferentes, por setas e movimento autônomo por rotas salvas.

Palavras-chave: Acessibilidade. Navegação autônoma. Cadeira de rodas. ROS.

ABSTRACT

The Brazilian resources of accessibility that are available for people are too far from the ideal. Almost 7% of Brazilian population was born with locomotion restrictions according to IBGE in 2010. Analyzing the scenario described, the goal of the project is to improve a motorized wheelchair giving it new features so that most of this audience is covered and can enjoy a transport that requires less effort. On software, the project used ROS, which provides a framework kit for robot’s development, and an Android app used with the system's interface. On hardware, as it will not be built as a physical prototype, the Gazebo simulator was used to show the behavior combined with the program. The hardware model was also described in case of a future physical build, and it contains engines, Raspberry microprocessor, Arduino, voltage regulators and LIDAR. Therefore, the project proposes to use different accessible technology to build a motorized wheelchair that gives a better mobility to the user, using a cellphone app that will allow him to use two different control options, for arrows and independent movement by saved routes.

Keywords: Accessibility. Autonomous navigation. Wheelchair. ROS.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

As pessoas cadeirantes enfrentam inúmeras dificuldades. As barreiras físicas e arquitetônicas são uma das mais presentes, estando em suas residências, no ambiente de trabalho e nas instituições de ensino, por isso a acessibilidade é um fator muito importante na relação entre sociedade e os indivíduos com deficiência.

Segundo o Censo de 2010, no gráfico de População residente por tipo de deficiência permanente do IBGE (Instituto Brasileiro de Geografia e Estatística), existem no Brasil cerca de 13,3 milhões de pessoas (6,96% da população) que apresenta algum tipo de deficiência motora, das quais mais de 4 milhões têm grande dificuldade ou não conseguem se locomover precisando do auxílio de uma cadeira de rodas. Considerando também que o número de idosos está aumentando mais a cada dia, há também essa faixa da população que possui maior probabilidade de perder a capacidade de locomoção.

Enxergando as dificuldades de pessoas que possuem alguma categoria de deficiência motora, há novas tecnologias que pode ajudar na melhoria de sua qualidade de vida, e esta condição deve ser sempre buscada, por isso o trabalho foi desenvolvido pensando na mobilidade de deficientes físicos cadeirantes, principalmente aqueles com mobilidade dos membros superiores reduzida, considerando também as suas inúmeras dificuldades sociais, econômicas e financeiras.

O projeto portanto propõe usar diferentes tecnologias acessíveis para criar uma cadeira de rodas motorizada que permita ao usuário ter uma melhor mobilidade, ao utilizar um aplicativo de celular que o possibilitará usar duas opções de controle diferentes, por setas e movimento autônomo por rotas salvas.

O software foi desenvolvido através do ROS (*Robot Operating System*), que tem a função de gerenciar a interação entre os componentes. Foi utilizado o Gazebo para a simulação, juntando o ambiente criado no próprio simulador e a cadeira modelada no blender para substituir a montagem física. Com isso, foi possível desenvolver o mapeamento e a navegação autônoma, além de executar os comandos advindos do aplicativo desenvolvido no sistema Android.

Apesar da implementação do hardware ser através de uma simulação, também é descrito como poderia ser feito fisicamente, descrevendo o circuito que contará com motores para movimentação da cadeira, o arduino como um controlador dos motores, um sensor lidar para identificar obstáculos e mapear o ambiente e um Raspberry para o processamento do ROS.

2. TRABALHOS RELACIONADOS

Tendo em vista as necessidades e tecnologias já citadas, existem muitas iniciativas para criar cadeiras autônomas pelo mundo, utilizando várias tecnologias e ideias diferentes, por isso, a seguir são descritas aquelas que possuem maior sinergia com este trabalho e que de alguma forma contribuíram com os conceitos e ideias que são demonstrados.

Na cadeira de Murarka et al. (2009) se discute sobre o foco no usuário, além de se aprofundar nas ideias de mapeamento local para ser usado na identificação de obstáculos e áreas de perigo. Nessa proposta as direções podiam ser dadas via joystick ou laptop. Na Figura 1 é demonstrada a cadeira utilizando um laptop para mover-se no espaço. Para isso, dentre outros sensores, foi utilizado um LIDAR para criar, atualizar um mapa e simultaneamente, detectar a posição da cadeira dentro dele.

Figura 1 - Cadeira de rodas inteligente de Murarka et al em teste

Fonte: Murarka et al., 2009

Com foco no ROS e também utilizando um sensor LIDAR, o trabalho de Grewal et al., LIDAR-Based Autonomous Wheelchair, de 2017, se preocupa em criar um sistema menos exigente para os usuários fazendo com que simples comandos possam fazer caminhos mais complexos e evitando obstáculos automaticamente. Na Figura 2 é demonstrada como nesta cadeira o movimento é executado, utilizando um joystick para o controle básico do usuário.

Figura 2 - Cadeira de rodas de Grewal et al. com usuário

Fonte: Grewal et al., 2017

Usando um sistema incorporado, a cadeira de Yook et al. (2018) é controlada por um aplicativo para Android desenvolvido e instalado em um *smartphone*. Para este projeto, o sistema foi dividido em dois modos principais: modo de reconhecimento de voz e modo de toque. Neste projeto foi utilizado um Arduino Uno para executar todos os comandos.

Mais semelhante aos projetos de Murarka et al (2009) e Grewal et al (2017), Oliveira et al. (2019) acrescentaram em sua cadeira, um Raspberry para o processamento do LIDAR e uma tela touch screen para fazer a navegação do usuário ao invés de utilizar um computador e o joystick e com isso conseguiram realizar uma melhoria no processo de implementação tornando-o mais acessível que os outros Projetos.

No mercado tivemos uma novidade, o modelo Ci, da empresa Whill cuja imagem pode ser observada na Figura 3, que possui uma roda com aderência, sensores que ajudam nas curvas e a desviar de obstáculos e pode ser controlada por Bluetooth, no lugar do joystick, com um auxílio de um aplicativo para iphone que também mostra a bateria disponível. Por se tratar de um item comercial maiores detalhes de sua tecnologia não são revelados ao público, porém este item ainda está em fase de teste em 2021, não estando disponível para pessoas sem mobilidade.

Figura 3 - Cadeira Model Ci da Whill

Fonte: Whill inc, 2021

De muitas formas as ideias de cadeiras inteligentes têm se desenvolvido, com ideias às vezes diferentes e às vezes semelhantes apenas adicionando alguma melhoria. Indo pelo segundo caminho esse projeto busca melhorar as ideias anteriormente descritas, utilizando o sistema do ROS e colocando o LIDAR que é um poderoso sensor, observamos que apesar disso os modos de controle dos projetos de Murarka et al. (2009), Grewal et al (2017), Oliveira et al. (2019), não são práticos no quesito de enviar os comandos ao sistema, pois apesar de ajudar com desvios e obstáculos necessitam de um grande controle do usuário ainda, se apegando ao uso do computador que não é nada prático ou de um joystick como já é feito nas cadeiras motorizadas tradicionais que podem trazer problemas e uma difícil substituição em caso de defeito.

Em contraponto a isso observamos nas cadeiras de Yook et al. (2018) e da Whill ótimos controles via smartphone, o que traz grande praticidade principalmente nos dias de hoje em que muitas pessoas possuem um celular e com isso o equipamento não sofre com o problema de difícil substituição em caso de defeito, entretanto, esses projetos não utilizam um sistema de mapeamento como o ROS, que fornece facilidades para projetar desvios de obstáculo e outras vantagens que esse sistema pode proporcionar, como a possibilidade que este projeto apresenta, de ter um mapa personalizável em que o usuário pode ir para pontos específicos, apenas dando um clique na tela.

3. FUNDAMENTOS TEÓRICOS

Nesta seção apresentaremos as principais ferramentas e conceitos que foram utilizados para o desenvolvimento deste projeto, dando-lhes descrições concisas e explicando seus principais usos.

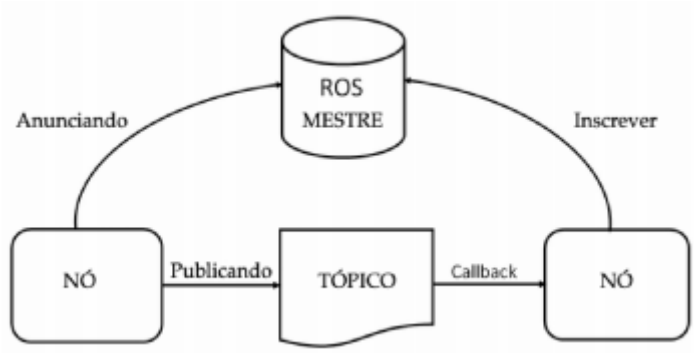
3.1. ROS

Para auxiliar no desenvolvimento do controle autônomo da cadeira de rodas, será utilizado o ROS que, segundo Quigley et al. (2015, p. 3), possui um conjunto de frameworks (pacotes de programas que promovem uma funcionalidade genérica) de código aberto para desenvolvimento de aplicações para robôs. O objetivo principal desse sistema é disponibilizar ferramentas e infraestrutura para construção de robôs de forma mais fácil, onde é possível desenvolver em meses o que levaria anos se fosse feito do zero. Com o advento do ROS, a robótica se tornou mais acessível a novos usuários, fornecendo uma plataforma onde pessoas podem compartilhar códigos e ideias de forma mais rápida. Na prática, o ROS (Robot Operation System) integra módulos em um robô que se comunicam trocando e processando dados, tais como câmera, sensores, GPS (Global Positioning System), CPU (Central Processing Unit) e etc.

Entre os diversos conceitos do ROS, destacam-se para o desenvolvimento de projetos: os nós, os tópicos e os serviços. Segundo Joseph (2018, p. 151), os nós são módulos que publicam (enviam) ou subscrevem (recebem) dados por meio de um canal de comunicação denominado de tópico ou que chama uma função que executa em outro nó através de um canal de comunicação denominado serviço. É assim que ocorre a integração em paralelo de todas as partes dos sistemas. Desta forma, o processamento dos cálculos e da imagem em um nó, e simultaneamente outro nó executa o controle dos motores do robô. Além disso, a comunicação de dados pode ser realizada com uso do protocolo TCP/IP, possibilitando o planejamento de tarefas à distância do robô, por exemplo, um dispositivo enviando mensagens como um Mestre.

Conforme a Figura 4, os nós podem se inscrever ao mestre para receber informações através de tópicos, e nós também podemos anunciar ao mestre para publicar informações através dos tópicos.

Figura 4 - Comunicação entre Nós



Fonte: Martignoni, 2016

3.1.1. ROSLAUNCH

O Roslaunch é uma das principais ferramentas do sistema ROS, pois inicializa de uma forma simples múltiplos nós e gerencia os respectivos parâmetros. Ele trabalha com arquivos no formato XML, com extensão .launch. Para executar no terminal linux, primeiro inserimos o comando roslaunch, precedido pelo nome do pacote que contém o arquivo launch e em seguida o nome do arquivo (ROS.ORG, 2020).

3.1.2. RVIZ

Uma ferramenta muito útil no ROS é o Rviz, que segundo (RETHINK ROBOTICS, 2015) é um visualizador 3D para exibir dados de sensores e informações de estado do ROS. Usando o RVIZ, é possível visualizar a configuração atual do robô em um modelo virtual. Também é possível exibir representações ao vivo de valores de sensores provenientes de tópicos.

3.1.3. SLAM

O termo SLAM significa *Simultaneous Localization And Mapping*. O objetivo desse processo é criar um mapa usando um robô ou veículo que navega no ambiente enquanto usa o mapa que ele gera.

Segundo (SOLÀ, 2014) o SLAM consiste em três operações básicas, que são reiteradas a cada etapa do tempo:

- O robô se move, alcançando um novo ponto de vista. Devido a ruído inevitável e erros, esse movimento aumenta a incerteza na localização do robô.
- O robô descobre recursos interessantes no ambiente, que precisam ser incorporados ao mapa. Chamamos esses recursos de marcos. Devido a erros nos sensores, a localização desses pontos de referência é incerta. Como a localização do robô já é incerta, essas

duas incertezas precisam ser adequadamente compostas. Uma solução automatizada requer um modelo matemático de observação inversa para determinar a posição dos pontos de referência na cena a partir dos dados obtidos pelos sensores.

- O robô observa pontos de referência previamente mapeados e os utiliza para corrigir sua auto localização e a localização de todos os pontos de referência no espaço. Nesse caso, portanto, as incertezas na localização e nos marcos diminuem.

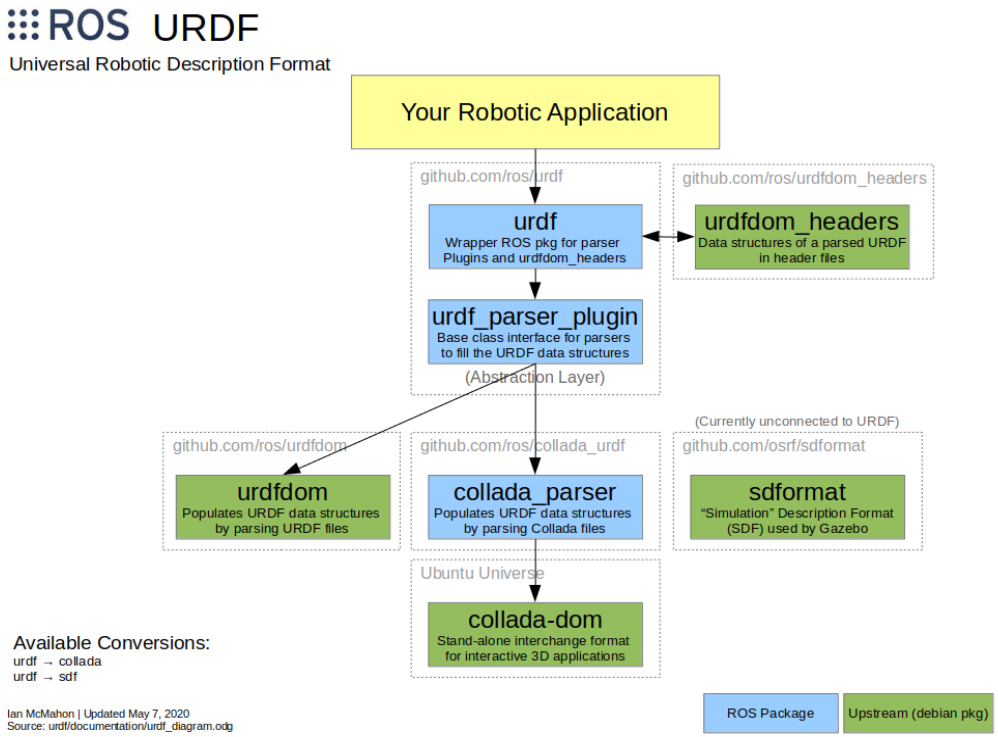
3.2. GAZEBO – SIMULADOR

O Gazebo é uma poderosa ferramenta de simulação que oferece recursos com precisão e eficiência para robôs em ambientes internos e externos complexos. Além de ser uma ferramenta gratuita e de código aberto, possui uma mecânica de análise física robusta, com interfaces gráficas programáveis, detalhadas e realistas. A simulação dos controladores de um robô no Gazebo pode ser realizada usando o *ros_control* e um simples adaptador de plug-in do Gazebo. Além disso, sua interface com o ROS é a mesma tanto para simulação quanto para um robô na realidade.

3.3. ARQUIVO URDF

Para representar um modelo 3D no simulador é necessário um arquivo XML no formato URDF (Unified Robotic Description Format), segundo Quigley et al. (2015, p. 281) este arquivo descreve as formas físicas e cinemáticas do robô e é representado por uma estrutura de árvore que contém basicamente elos e juntas, onde no elo se encontra as descrições do visual, inércia, material e colisão, reunindo a posição, tipo e hierarquia de cada objeto. Na Figura 5 é possível observar a estrutura do URDF.

Figura 5 - Representação em árvore do arquivo URDF



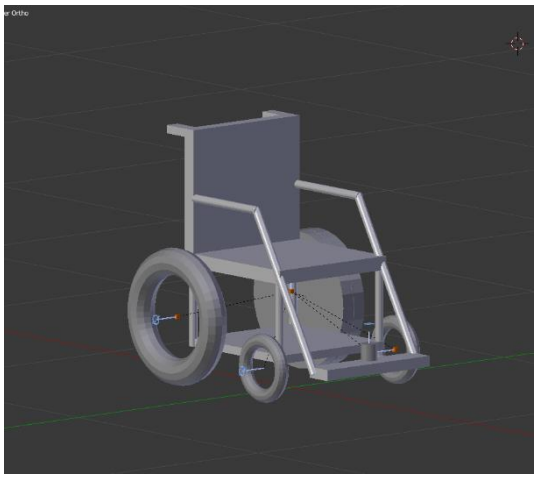
Fonte: Wiki ROS, 2019

3.3.1. Visual

Descreve o visual que o robô será apresentado no simulador, podendo tomar como forma geométrica um cubo, cilindro, esfera ou mesh (malha) este último é um arquivo de representação de um objeto mais elaborado portanto é adicionado um arquivo em formato .dae (Collada) ou .stl (StereoLitography) e há também a possibilidade de determinar a escalo do objeto geométrico.

No visual, determina-se a origem do objeto em relação ao elo nas coordenadas xyz e ângulo de rotação, arfagem e guinada, como também cor e textura do material. Exemplo na Figura 6.

Figura 6 - Cadeira de rodas modelada



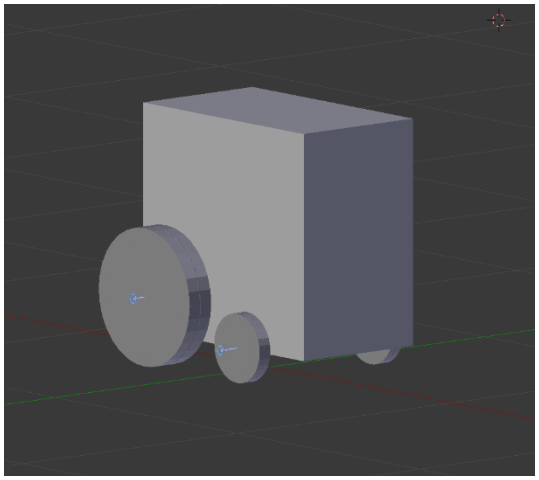
Fonte: autor

3.3.2. Colisão

O elemento de colisão é uma tag do código para a interação com a simulação, um subelemento direto do objeto de link, no mesmo nível da tag visual, a forma de visualização no simulador é a mais simples possível, conforme exemplificado na Figura 7, pois desta forma facilita o processamento.

No código de colisão é determinado as coordenadas da origem do objeto geométrico em relação ao elo e as coordenadas de rotação igual ao visual.

Figura 7 - Visualização do módulo de colisão



Fonte: autor

3.3.3. Inércia

Assim como a colisão, a Inércia do robô é de extrema importância para que ocorra uma simulação correta; Esta tag do URDF é sem dúvidas uma das mais importantes pois há uma base teórica sobre Momentos de Inércia. O momento de inércia, também conhecido como

inércia rotacional ou como massa angular, mede a resistência de um objeto à aceleração rotacional em torno de um eixo específico. Também definimos a massa no código, pois o tensor de inércia depende tanto da massa quanto da distribuição de massa do objeto (MORIN, 2008).

No código da inércia determinamos as coordenadas em relação ao elo e sobre as coordenadas rotacionais que são representadas por uma matriz 3x3.

Apesar de descrever as condições físicas do robô ainda há a necessidade de um arquivo a parte para complemento do URDF, que contém informações de simulação como por exemplo coeficientes de atrito (mu1/mu2), rigidez de contato (kp) e amortecimento(kd), velocidade de correção máxima de contato (maxVel) entre outros.

3.4. PACOTE TF

Conforme descrito na seção do arquivo URDF, um robô é composto por links e joints. Em ambos estão definidos frames para possibilitar a localização de todas as partes do robô no espaço. Logo é de extrema importância realizar o gerenciamento das transformações entre frames. Para essa tarefa é utilizado o pacote TF, um dos pacotes mais importantes para navegação e localização. O gerenciamento adota uma abordagem distribuída, utilizando tópicos para compartilhar os dados de transformações. Ou seja, não existe um servidor central de informações de transformações, sendo assim qualquer nó pode publicar ou subscrever os dados de transformações. Sem o pacote TF, o usuário teria que definir todas as matrizes de transformações manualmente, o que aumentaria o grau de complexidade do processo (QUIGLEY et al., 2015).

O formato de mensagem utilizado no pacote TF é o tf/tfMessage, que contém uma lista de transformações, especificando as dependências (parent e child) dos frames envolvidos, com suas posições e orientações (QUIGLEY et al., 2015).

3.5. NAVIGATION STACK

Segundo Quigley et al. (2015), para desenvolver navegação autônoma para um robô a partir de um mapa conhecido, três nós são de extrema importância: map_server, amcl e move_base.

3.5.1. Map_server

O nó map_server faz parte do pacote map_server. Ele fornece o mapa estático no qual o robô irá se localizar e planejar a navegação. Para tal, é necessário fornecer o mapa estático

em arquivo yaml (Yet Another Markup Language) pois por meio deste que os nós podem receber parâmetros, por se tratar de uma linguagem orientada a dados de serialização hierárquica legível para o programador, mais simples que as demais linguagens, pois não propõe marcações e tags e tem como objetivo o armazenamento de dados de configuração (QUIGLEY et al., 2015; ROS.ORG, 2020).

Além do nó, o pacote map_server fornece o utilitário de linha de comando map_saver, que permite que mapas gerados dinamicamente sejam salvos em arquivos, sendo um pgm e um yaml. O arquivo pgm contém o mapa. Já o yaml, contém os metadados do mapa, como o path do arquivo pgm, a resolução do mapa, a localização da origem, os limites para espaços livres e ocupados (QUIGLEY et al., 2015; ROS.ORG, 2020).

3.5.2. Amcl

O nó amcl faz parte do pacote amcl. Ele é responsável pela localização do robô utilizando o mapa, fornecido pelo nó map_server através do serviço static_map (nav_msgs/GetMap). Ele implementa um conjunto de algoritmos de localização probabilística, conhecidos coletivamente como Localização de Monte Carlos Adaptativa (QUIGLEY et al., 2015; ROS.ORG, 2020).

Para a localização do robô, utilizamos como conceito a pose, que compreende a posição e a orientação em relação ao frame do mapa. O amcl mantém uma distribuição de poses possíveis onde o robô possa estar e atribui probabilidades de cada uma delas representar a pose real do robô. Conforme o robô se movimenta e, consequentemente, os sensores realizam as medições, os dados são comparados com aqueles que seriam esperados para cada uma das poses seguindo as estimativas de odometria, e a probabilidade de cada pose é atualizada. Com o tempo, as poses com alta probabilidade permanecem, e acabam convergindo para a pose com a maior probabilidade (muito provavelmente não seja a pose real do robô, no entanto é precisa o suficiente para a navegação) (QUIGLEY et al., 2015).

3.5.3. Move_base

O nó move_base faz parte do pacote move_base. Ele é responsável pelo planejamento do caminho, e é dividido em dois módulos: o global path planner e o local path planner. O global path planner é responsável por gerar um caminho global livre de obstáculos da posição inicial até o objetivo, utilizando como base o mapa do ambiente. Já o local path planner tenta seguir o global path, utilizando odometria e as leituras feitas pelos sensores para evitar os obstáculos (KOUBAA, 2016).

Outro conceito importante é o de costmap, que são utilizados para armazenar informações de obstáculos no mapa. São utilizados dois tipos de costmap: o local_costmap e o global_costmap (as configurações em comum podem ser definidas em um costmap único para ambos) (KOUBAA, 2016).

O local path planner é responsável por enviar os comandos de velocidade por meio do tópico cmd_vel. O tipo de mensagem publicada é geometry_msgs/Twist, que é composta por dois tipos de sub mensagens: linear e angular. A linear é usada para os componentes da velocidade linear em metros por segundo. Já a angular é usada para os componentes da velocidade angular em radianos por segundo (KOUBAA, 2016).

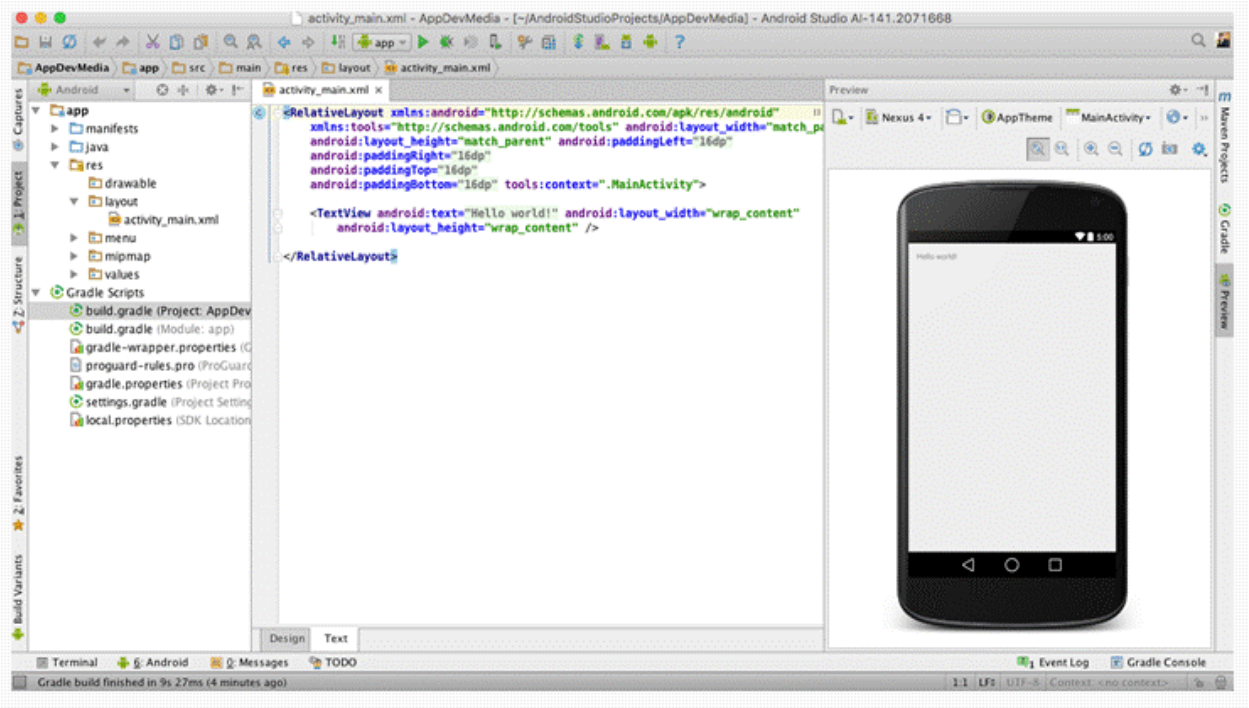
Para o planejamento do caminho local, o ROS fornece um algoritmo chamado Dynamic Window Approach (DWA), que primeiro mostra discretamente o espaço de controle e, em seguida, executa simulações e seleciona o comando de velocidade mais eficiente (KOUBAA, 2016).

3.6. IDE – ANDROID STUDIO

A IDE escolhida para o desenvolvimento da aplicação Android foi o Android Studio, demonstrado na Figura 8, que é um ambiente de desenvolvimento integrado (IDE da sigla em inglês, Integrated Development Environment) oficial do Google para a construção de aplicativos móveis.

Baseado no IntelliJ IDEA, o Android Studio é um ambiente de desenvolvimento muito rico em recursos para aumentar a produtividade durante o processo de trabalho em apps. Tendo também uma variedade imensa de customização, que se torna extremamente necessária para quem já está familiarizado com outras IDEs, sendo possível: personalizar o tema geral da apresentação, editar os atalhos do teclado, Frameworks e ferramentas de testes com centenas de possibilidades, ferramentas para análise de desempenho, um emulador de celular extremamente eficiente para que simule sua aplicação em tempo real abrindo possibilidades para simular em diferentes tipos de celulares, entre outros recursos (DEVMEDIA, 2016).

Figura 8 - Interface do Android Studio



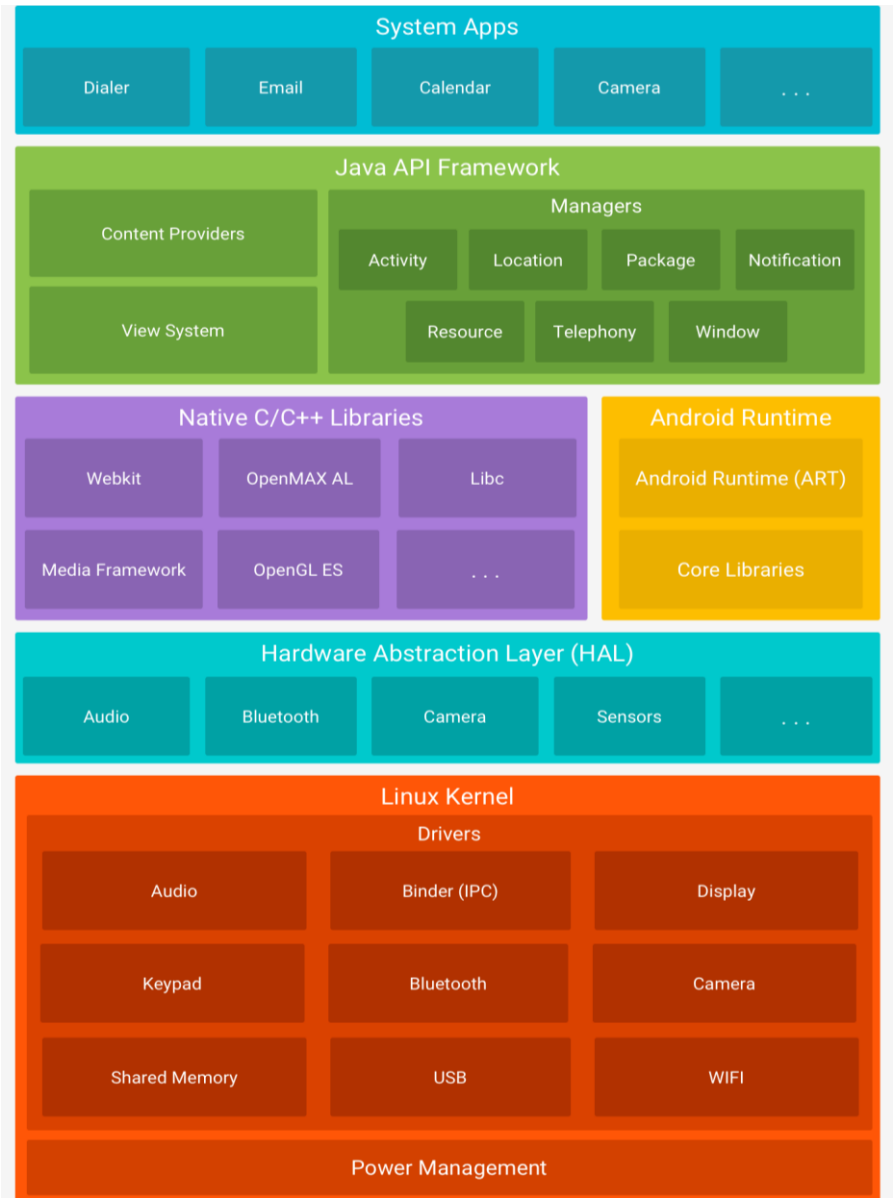
Fonte: Devmedia, 2016

3.7. ARQUITETURA ANDROID

O Android é um sistema operacional baseado no Kernel Linux desenvolvido majoritariamente pelo Google. De início o principal foco eram os dispositivos móveis com tela sensível ao toque (smartphones e tablets), porém a tecnologia evoluiu de uma forma que hoje é possível encontrar uma série de dispositivos Androids como televisões, relógios, videogames, câmeras digitais e até mesmo automóveis (DEVELOPERS, 2020).

Para entender um pouco melhor como é o funcionamento deste Sistema Operacional, analisada a arquitetura, como pode ser observado na Figura 9.

Figura 9 - Arquitetura do Sistema Android



Fonte: Developers, 2020

A base de todo o Android é o Kernel Linux, que faz todo o gerenciamento baixo nível de integração com Hardware e aproveita os principais recursos de segurança onde os fabricantes de dispositivos possam desenvolver drivers seguros para um Kernel conhecido (DEVELOPERS, 2020).

Acima do Kernel temos a Camada de Abstração de Hardware (HAL da sigla em inglês, Hardware Abstraction Layer) que, como o nome já diz, faz a abstração de funcionalidades e capacidades do hardware para a API de alto nível, que é onde os desenvolvedores constroem suas aplicações. Através da HAL é possível carregar o módulo da biblioteca para o hardware, sendo possível acessar componentes como câmera, Bluetooth etc (DEVELOPERS, 2020).

Logo em seguida da HAL temos a divisão de componentes que permite executar aplicações e os serviços do sistema e bibliotecas nativas. A Android Runtime, ou ART, é responsável pela compilação de arquivos do tipo DEX (Dalvik Executable, no qual encontramos um tipo de bytecode otimizado para baixo consumo de memória) em executáveis criados para o dispositivo durante a instalação de um aplicativo. Também possui um Garbage Collector para encerrar processos que não estão sendo mais usados, ajudando em uma execução rápida e processamento paralelizado (DEVELOPERS, 2020).

Chegando mais acima temos a API Java do Android, onde os componentes necessários para o desenvolvimento de apps ficam. Esta API de alto nível é responsável pelo gerenciamento de recursos, notificações, navegação entre apps, compartilhamento de dados e etc. É através daqui que o desenvolvedor possui as mesmas funcionalidades de aplicativos do sistema, podendo terceirizar aplicações como teclado, por exemplo. Além de também poderem ser substituídas através dos apps (DEVELOPERS, 2020).

Por fim, e não menos importante, vale ressaltar que o Sistema Operacional Android possui diversos níveis de API. À medida que a tecnologia avança e as versões vão crescendo, novas melhorias são publicadas de acordo com os pré-requisitos de hardware da maioria dos dispositivos atuais, ou seja, aparelhos um pouco mais antigos vão ficando obsoletos devido a este avanço. Portanto, é importante que o desenvolvedor leve isso em consideração para que possa abranger um maior número de pessoas com o seu serviço (DEVELOPERS, 2020).

3.8. DESENVOLVIMENTO ANDROID – JAVA

A Linguagem Java é uma linguagem de programação orientada a objetos desenvolvida na década de 90 pela Sun Microsystems e foi obtida pela Oracle Corporation em 2008. Java é extremamente conhecido pela sua arquitetura diferente, onde ao invés de ser compilado para um código nativo, é compilado para bytecode e é interpretado por uma máquina virtual Java (JVM da sigla em inglês, Java Virtual Machine). Graças a essa arquitetura a plataforma funciona de forma independente, onde desde que se tenha um ambiente Java é possível executar qualquer código escrito na linguagem. Na Figura 10 temos um exemplo prático (MACHADO, 2018).

Como dito anteriormente, Java é uma linguagem orientada a objetos. Isso quer dizer que é baseada na modelagem de um objeto e na comunicação entre eles. Na prática, criamos uma classe com as características básicas deste objeto, que chamamos de atributos. Os atributos podem ser como o tamanho de uma televisão, a cor de um carro ou até a quantidade de elétrons

em um átomo. Dentro dessas classes também temos os métodos ações que este objeto pode exercer, baseado nas lógicas de programação (MACHADO, 2018).

Ao instanciar um objeto (atividade de criar um objeto) a partir de uma classe que define o tal, fazemos referência a uma posição de memória com cada atributo que podem ser diferentes para cada um, e os métodos, este sendo iguais a não ser que haja uma sobrecarga dos mesmos (mas isso depende do tipo de classe que está trabalhando) (MACHADO, 2018).

Figura 10 - Exemplo, código de programação

```
1  /**
2   * Exemplo de uma classe simples em Java.
3   */
4  public class PrimeiraClasse {
5      public static void main(String[] args) {
6          System.out.println("Hello world !!!");
7      }
8  }
```

Fonte: Universidade Java, 2011

Para programação Android a sintaxe da linguagem permanece a mesma, porém a arquitetura muda um pouco se comparado a um projeto Java Backend ou até alguma aplicação de interface visual. Ao criar um projeto no Android Studio, é gerado um arquivo XML que faz referência ao código, o próprio código padrão e um arquivo chamado Manifest.xml como pode ser observado na Figura 11 que é o mandatório para qualquer aplicação Android contendo as informações de permissões, Activities e informações relevantes para a aplicação (DEVMEDIA, 2016).

Figura 11 - Manifest.xml gerado por padrão

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="br.com.devmedia.appdevmedia" >

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Fonte: Devmedia, 2016

Para o desenvolvimento também se conta com uma ferramenta gráfica muito útil onde pode-se adicionar botões, barras, etc., enquanto eles são importados automaticamente para o seu código sendo possível adicionar ações para quando eles são acionados. Nesta ferramenta também é possível definir layouts do aplicativo, adicionar novas telas, fixar objetos na tela e até configurar para caso haja uma rotação de tela.

3.9. SOCKET

Socket é um tipo de comunicação entre processos (IPC, da sigla em inglês Inter-Process Communication) que permite a troca de informação de forma bidirecional entre dois processos independentes, estando ou não na mesma máquina, através da rede. Para entender melhor onde a comunicação por socket se encaixa, precisamos falar um pouco sobre o comparativo do modelo OSI (que especifica um padrão de protocolos de rede) e a pilha TCP/IP que é usada como base para a aplicabilidade nas intranets e na internet (PANTUZA, 2017).

O modelo OSI foi criado há aproximadamente 40 anos, onde foi referenciado pela ISO como um padrão para protocolos de comunicação de redes computacionais para facilitar a interoperabilidade entre fornecedores diferentes. O modelo OSI é usado até hoje como

referência teórica, pois na prática é majoritariamente usado a pilha TCP/IP (PANTUZA, 2017).

A Figura 12 mostra um comparativo entre o modelo OSI com suas 7 camadas e a pilha TCP/IP com suas 4 camadas.

Figura 12 - Comparativo entre o modelo OSI e a pilha TCP/IP

Fonte: Pantuza, 2017

Tomando como base a pilha TCP/IP, o socket se localiza entre a camada de transporte e aplicação, pois é ele quem recebe ou transmite os dados referentes à aplicação para a rede para que assim possa se comunicar com outro dispositivo ou aplicação (PANTUZA, 2017).

Para que duas aplicações estabeleçam uma comunicação através de socket é necessário especificar que mais comumente é usada uma arquitetura de servidor/cliente. Resumidamente, o servidor é o dispositivo que disponibiliza as informações e serviços para outros ligados em rede, estando sempre disponível para conexão. Já o cliente é quem solicita e consome estes serviços (PANTUZA, 2017).

Este IPC possui diversas formas de comunicação (protocolos), porém duas ganham mais destaque por serem mais conhecidas, sendo elas o TCP e UDP (PANTUZA, 2017).

3.9.1. TCP

- Orientado à conexão (só transmite dados após estabelecer uma conexão e confirmá-la);
- Full-duplex, permite que as duas aplicações recebam e transmitam dados simultaneamente;
- Garante a entrega de dados sequencialmente;
- Garante a divisão de informação em pequenos pacotes para não sobrecarregar a transmissão (PANTUZA, 2017).

3.9.2. UDP

- Não é orientado à conexão;
- Não é full-duplex, ou seja, a comunicação é unidirecional;
- A divisão de dados deve ser feita manualmente (em datagramas);
- Não garante a entrega de dados, e muito menos de forma sequencial;

Por ser um protocolo extremamente mais simples, o diferencial do UDP se dá pela alta velocidade se comparado ao UDP, porém um pouco mais trabalhoso de ser implementado (PANTUZA, 2017).

3.10. SHELL SCRIPT

Shell script é o nome dado a uma linguagem de programação que automatiza tarefas através de scripts (algoritmos projetados para realizar determinadas tarefas) que será interpretado por algum interpretador tipo Shell (linha de comando do Linux ou UNIX, normalmente bash ou sh) (DEVMEDIA, 2015).

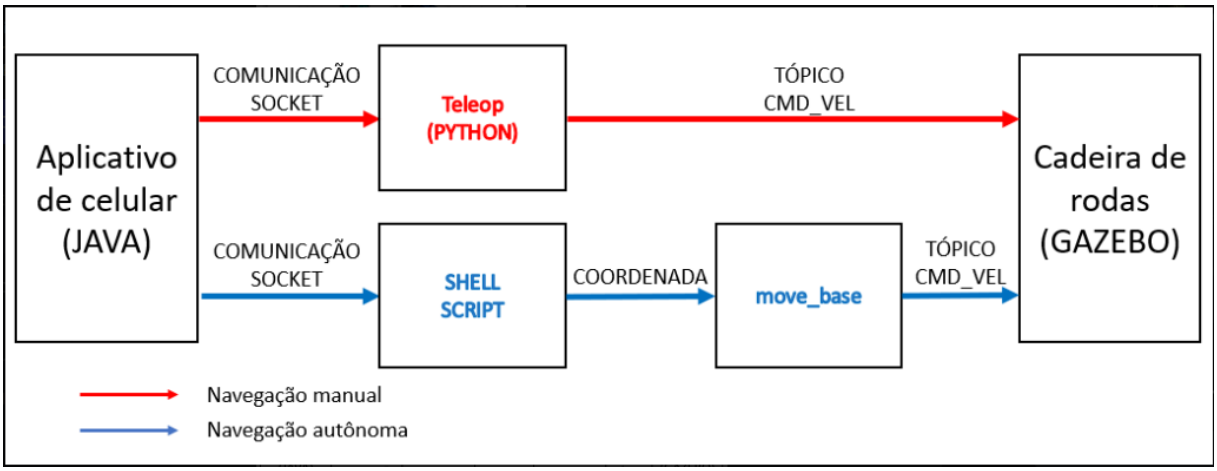
Geralmente o Shell script é utilizado para facilitar e automatizar operações rotineiras ou executar uma sequência de comandos para ganhar tempo em determinada atividade. Os exemplos mais comuns de uso são: manipulação de arquivos e diretórios, execução de programas, envio e recebimento de dados, impressões de texto etc (DEVMEDIA, 2015).

A utilização de Shell script pode parecer bem limitada por ser apenas uma sequência de comandos específicos, porém há muitas utilidades inclusive mais complexas, pois é possível utilizar diversas lógicas de programação vistas em outras linguagens como condições e loops (DEVMEDIA, 2015).

4. DESENVOLVIMENTO

Neste tópico será descrito como foram desenvolvidas todas as partes do projeto assim como é feita a conexão entre essas partes. A figura 13 apresenta um diagrama geral onde mostra o conjunto e interfaciamento dos sistemas. Em vermelho está representado o fluxo de informações quando trabalhando na navegação manual, os comandos são enviados pelo aplicativo para o nó Teleop atraves de comunicação *socket*, esse por sua vez trata as informações e envia os comandos de velocidade para a cadeira de rodas por meio do tópico *cmd_vel*. Já em azul está representado o fluxo de informações quando funcionando em navegação autônoma, os comandos também são enviados pelo aplicativo via comunicação *socket* para o *Shell script* que por sua vez interpreta os dados e envia as coordenadas para o nó *move_base*, que planejará a rota e enviará os comandos de velocidade através do tópico *cmd_vel*.

Figura 13 – Diagrama de funcionamento geral



Fonte: autor

4.1. GAZEBO - CRIAÇÃO DO AMBIENTE

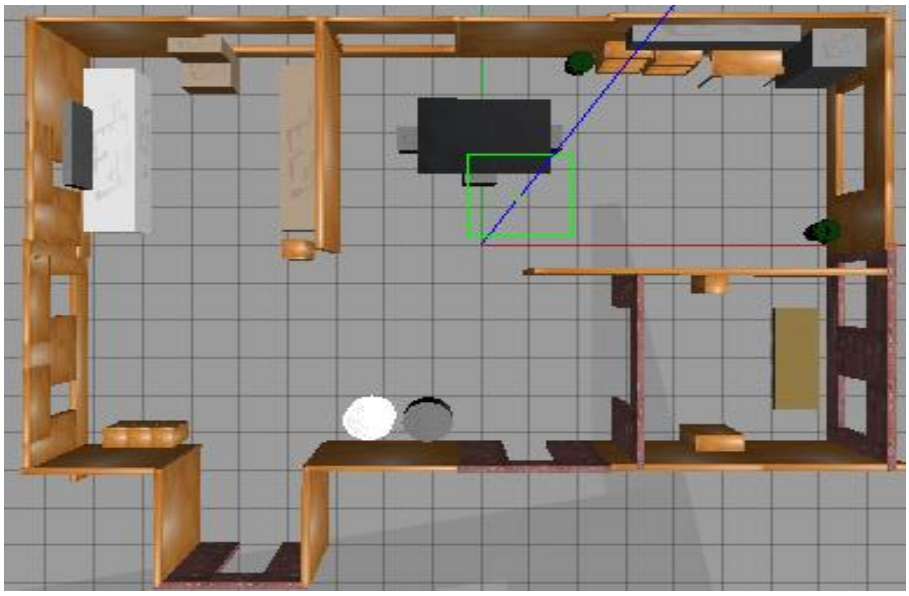
Como este projeto foi realizado de forma virtual, a primeira etapa a ser realizada foi criar um ambiente de simulação que pudesse demonstrar como a cadeira se portaria em uma residência, local para onde o projeto foi idealizado, para isso selecionamos o Gazebo.

No Gazebo encontramos duas opções para realizar essa tarefa, a primeira seria montar o ambiente com vários modelos de objetos disponibilizados pelo próprio simulador, a outra opção, que foi a escolhida, é de criar um modelo que seja a casa completa e colocar esse modelo no ambiente diretamente.

Para criação do modelo da casa, foi selecionado um bloco qualquer e colocado no

ambiente, selecionando a opção de editar modelo entramos no módulo de edição no Gazebo, com isso foram selecionados blocos e cilindros básicos, esses foram modificados para criar as paredes e móveis conforme tamanho e espessura, além das cores de cada um. Após finalizar o modelo completo, ele foi salvo em um novo arquivo chamado model.sdf e depois foi conectado no ambiente que foi chamado de DFRR_house.world, o resultado pode ser observado na planta demonstrada na Figura 14.

Figura 14 - Planta da casa na simulação



Fonte: autor

4.2. BLENDER – CRIAÇÃO DO ROBÔ

Para que haja a simulação no Gazebo é necessário um código com a descrição do robô, tem diversos tipos de arquivos de descrição, porém o mais difundido entre usuários do sistema ROS é o XML do tipo URDF.

Há diversas formas de construir o robô, como o arquivo é um código em linguagem Python pode-se escrever manualmente, porém o programador deve ter em mente exatamente as coordenadas de cada objeto e se na construção do robô houver uma peça mais detalhada deve incluir uma malha, o que torna a tarefa de construir o robô muito difícil de ser executada apenas usando códigos.

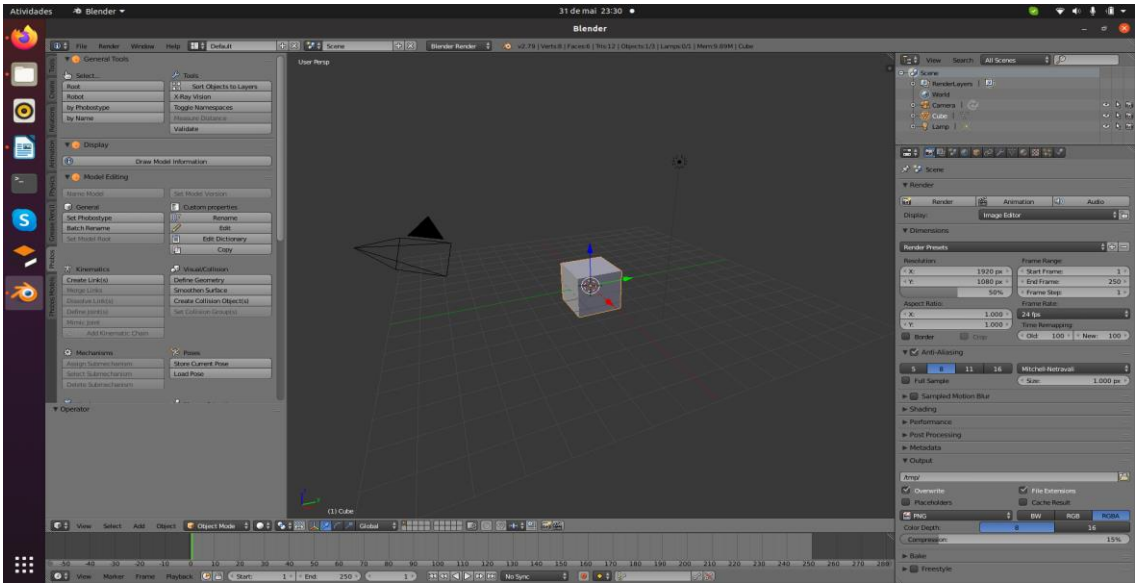
Outra forma de criar o URDF é utilizando programas de modelagem 3D que geram o código automaticamente. A princípio utilizamos o Free CAD, software de modelagem Open Source (Código Aberto) de fácil entendimento, porém após se familiarizar com o programa foi

descoberto que não havia a opção de exportar o arquivo no modelo URDF apenas arquivos STL e Dae que são utilizados nas criações das malhas. Outro software que também gera o URDF a partir de um plugin instalado é Solidworks que oferece a possibilidade de determinar eixos, juntas, colisão, inércia e o tipo de material utilizado, entretanto este programa não é acessível, por isso foi utilizado o Blender, que é um modelador, criador de animações, com renderização de código aberto que, em conjunto com um plugin de nome Phobos criado pelo centro de pesquisa de inteligência artificial da Alemanha, fornece uma forma simples de criação de robôs e ambientes.

Após baixar e instalar o Blender e o Photos, deve-se adicionar o Phobos no Workspace do Blender para que apresente as opções de construção do modelo, como demonstrado na Figura 15, onde se apresenta as seguintes opções:

- General – Caracteriza o tipo de objeto que for selecionado (visual, link, inércia, colisão entre outros)
- Custom properties – Edição do nome do objeto.
- Visual/Collision - Define o objeto que será descrito na tag colisão e cria o objeto
- Masses & Inertial – Determina as coordenadas de inércia na tag a partir de uma massa declarada
- Hardware – Define o objeto selecionado como um motor ou sensor
- Mechanisms - Define um grupo de objetos como mecanismos
- Kinematics – Este ícone cria as juntas e determina a hierarquia de cada objeto.

Figura 15 - Exemplo página de criação do Blender



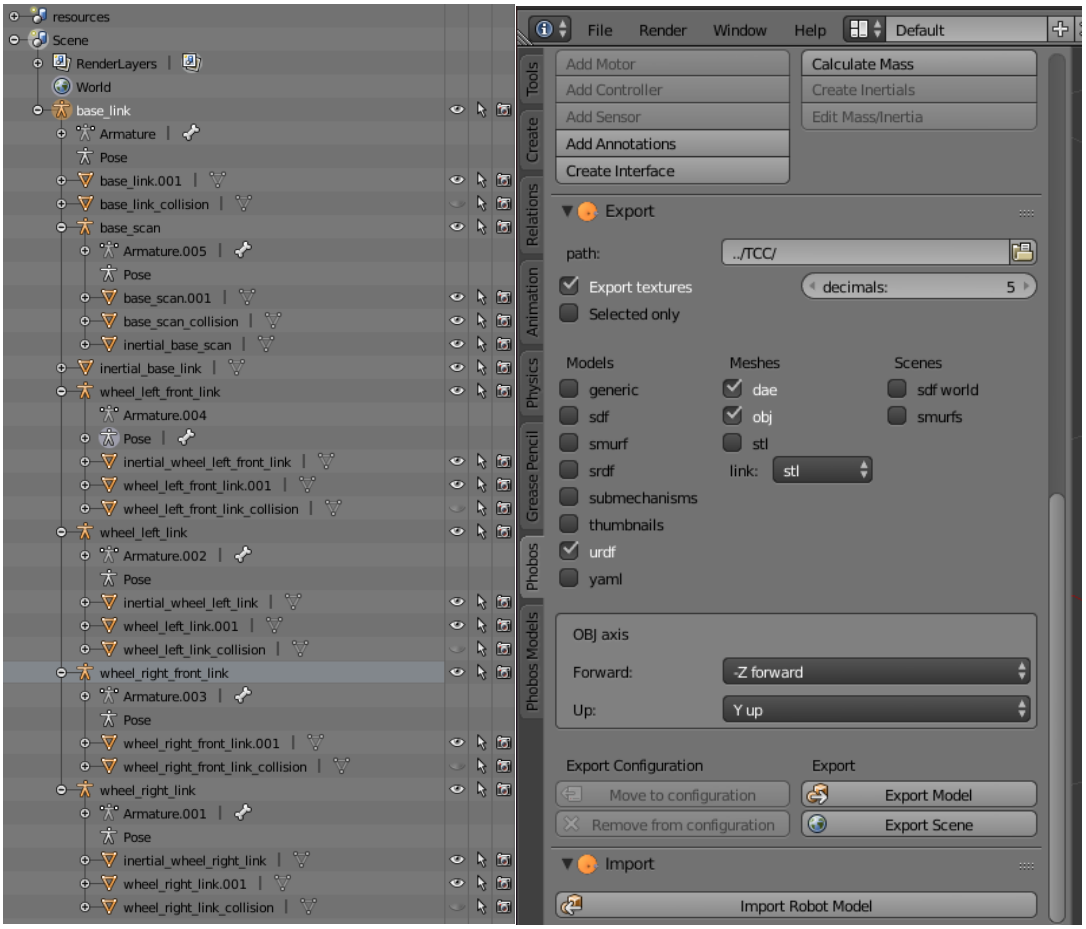
Fonte: autor

Além destas ferramentas no *Workspace* há a possibilidade de determinar cores e texturas ao modelo estas opções são aplicadas com as ferramentas do *Blender*, porém no momento de exportar para URDF as informações ficam registradas.

No próprio *Blender* se apresenta a hierarquia dos elementos, fornecendo liberdade para modificar a posição, o que facilita se o modelo construído for complexo, na Figura 16 está demonstrada a árvore do projeto *wheelchair*. Após construir o robô e adicionar todos os parâmetros necessários, então deve-se exportar no formato URDF, escolhendo o formato de malhas com STL ou Dae.

O resultado da Cadeira pode ser obsevado na figura 17.

Figura 16 - Configurações Blender



Fonte: autor

Figura 17 - Resultado da cadeira feita no Blender

Fonte: autor

4.3. AMBIENTE DE SIMULAÇÃO

Antes de iniciar a simulação da cadeira no gazebo é preciso preparar alguns arquivos na pasta de simulações do ROS, indo até o local dos arquivos de *launch* criamos o wheelchair_house.launch, no pacote wheelchair_gazebo, no qual se inclui outro *launch* denominado empty_world.launch presente no pacote gazebo_ros, que apresenta todas as funcionalidades que necessitamos, sendo necessario alterar apenas parâmetro world_name, substituindo o arquivo do mundo empty.world pelo arquivo DFRR_house.world, que contém o ambiente da casa. Além desse arquivo launch, ele inicializa um script python chamado spawn_model, localizado no pacote gazebo_ros, que inclui a cadeira de rodas no Gazebo através do arquivo URDF da mesma.

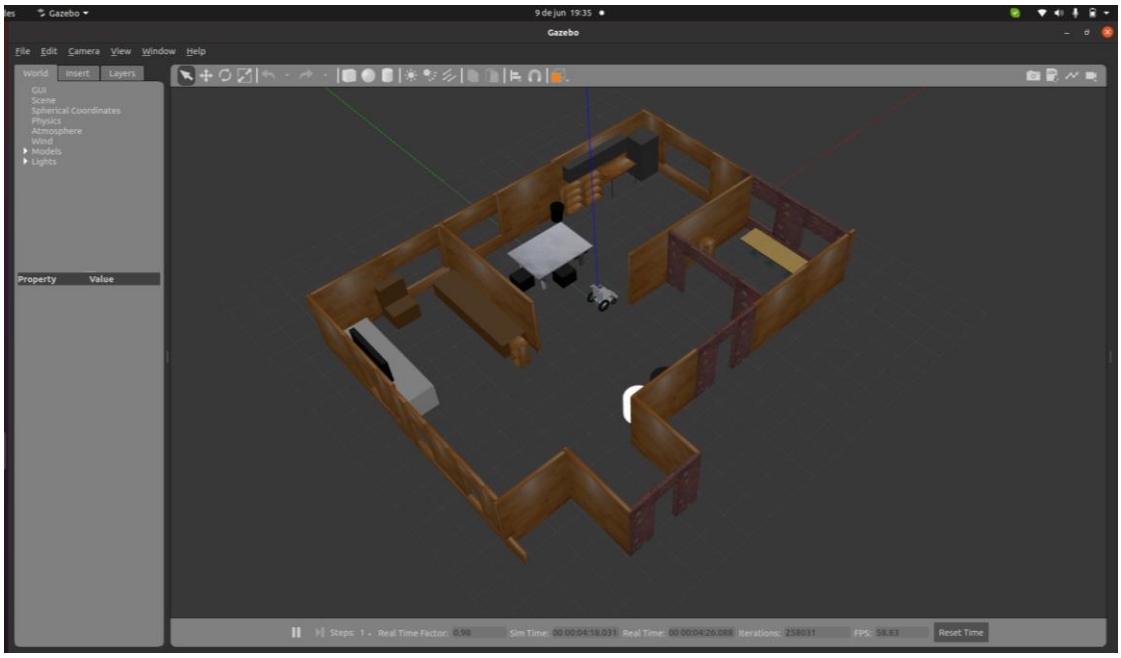
Com o ambiente e a cadeira criados, o primeiro passo para a simulação é iniciar o Gazebo com os modelos. Para isso, é necessário digitar o comando da Figura 18 no terminal linux e com isso um ambiente semelhante ao da Figura 19 será aberto.

Figura 18 - Código Simulação dos modelos

```
roslaunch wheelchair_gazebo wheelchair_house.launch
```

Fonte: autor

Figura 19 - Ambiente Gazebo com a cadeira



Fonte: autor

4.4. MAPEAMENTO

Para que posteriormente esse ambiente seja navegado automaticamente pelo robô, que neste caso é a cadeira, este necessita ser mapeado. Para isso, o ROS possui o pacote gmapping, que a partir do nó slam_gmapping fornece o SLAM baseado no LIDAR. Para tal, a cadeira deve navegar de forma manual pelo ambiente e através da leitura do sensor, esta ferramenta fará o mapeamento, criando uma imagem com extensão pgm, semelhante a que foi criada neste projeto e que pode ser observada na Figura 21, e um arquivo yaml para a conexão posterior com os outros nós do projeto.

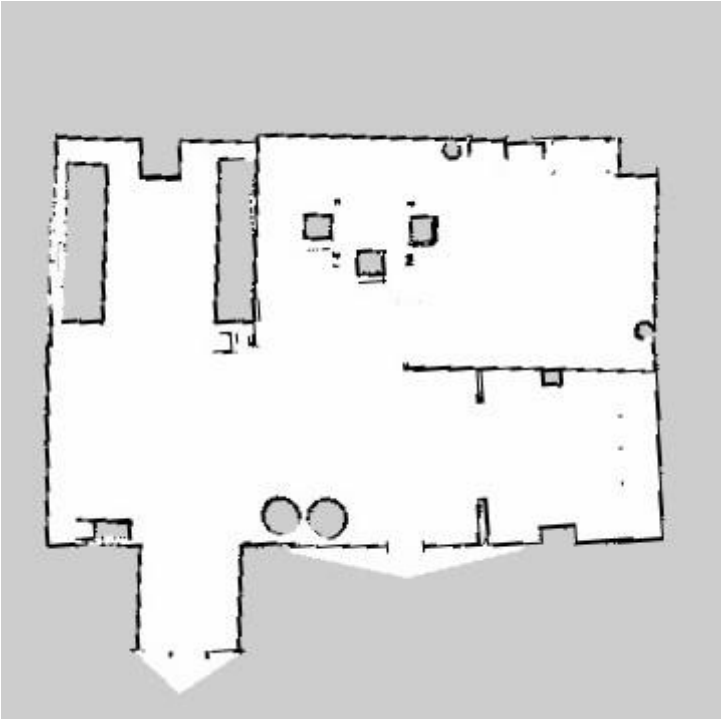
Para iniciar o SLAM se usa o código exemplificado na Figura 20.

Figura 20 - Código para iniciar o SLAM

```
roslaunch wheelchair_slam wheelchair_slam.launch slam_methods:=gmapping
```

Fonte: autor

Figura 21 - Mapeamento do ambiente

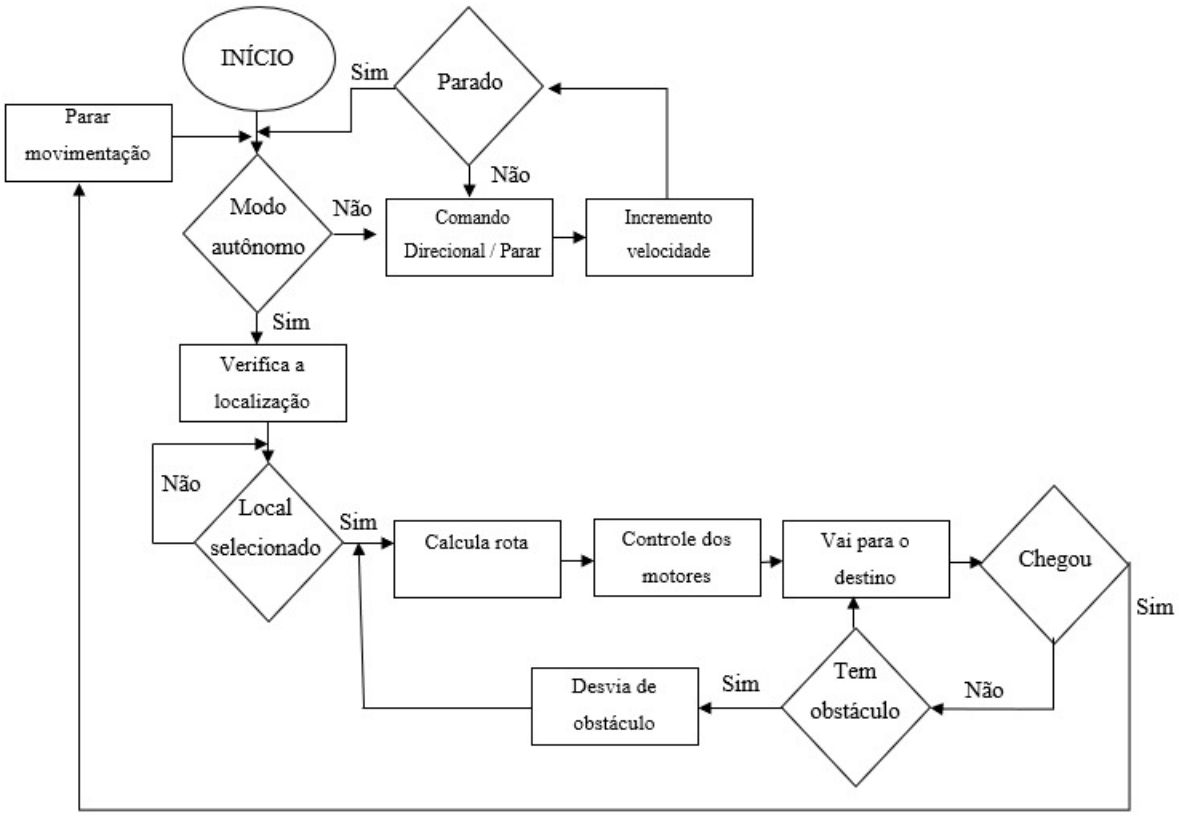


Fonte: autor

4.5. NAVEGAÇÃO

Após inicializar o ambiente e mapeá-lo, começa o processo mais prático, onde será simulado o que o usuário final deverá executar, para isso teremos dois módulos: teste de navegação pelo teleop e o teste de navegação pelo move_base. A fim de explicar o funcionamento geral a Figura 22 apresenta o fluxograma de decisões que a cadeira deverá tomar.

Figura 22 - Fluxograma de decisões



Fonte: autor

4.5.1. Teste de navegação pelo Teleop

Para a navegação por setas, foi desenvolvido um código em *Python* denominado *wheelchair_teleop.py*, localizado no pacote *wheelchair_teleop*. O código recebe comandos de velocidade do aplicativo através do socket, sendo que pode ser linear (setas para cima e para baixo) e angular (setas para esquerda e para direita). Além desses botões, há o de parar, que faz com que o robô pare instantaneamente. Nesse código, podemos definir as velocidades máximas e mínimas e o passo (menor velocidade possível), tanto para o linear quanto para o angular.

4.5.2. Teste de navegação pelo move_base (autônomo)

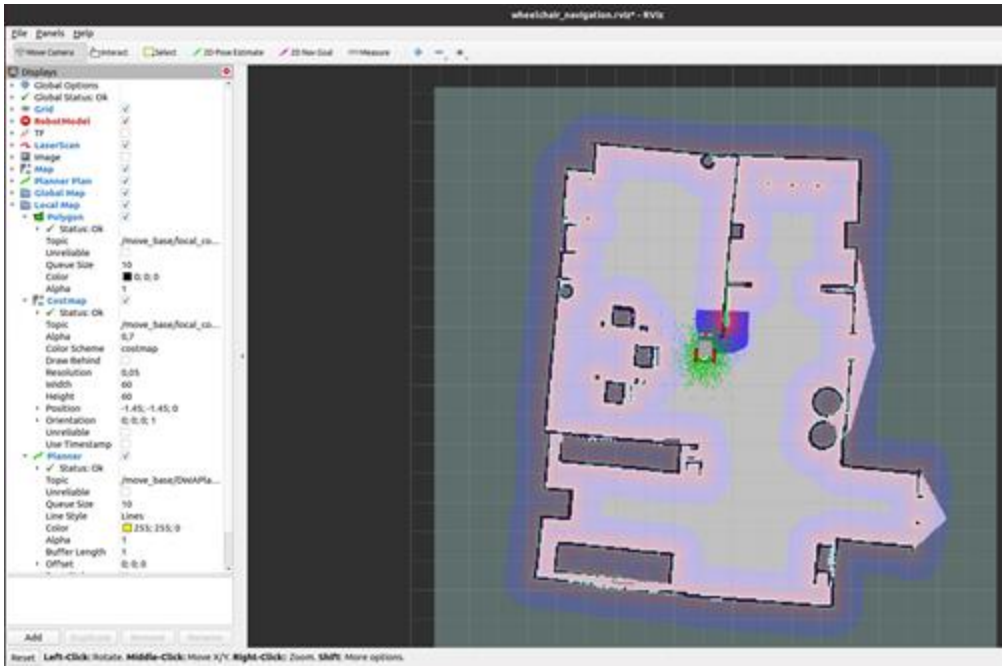
Para a implementação da navegação autônoma, foi criado o pacote *wheelchair_navigation* e na respectiva pasta para os arquivos launch foi gerado o arquivo *wheelchair_navigation.launch*. Esse arquivo é responsável pela a inicialização dos nós *robot_state_publisher*, *map_server*, *amcl*, *move_base* e rviz. O comando para executar o sistema de navegação autônomo da cadeira no terminal Linux é apresentado na Figura 23. Um exemplo de janela do Rviz que é aberta pode ser visto na Figura 24.

Figura 23 - Comando para iniciar o Rviz

```
roslaunch wheelchair_navigation wheelchair_navigation.launch
```

Fonte: autor

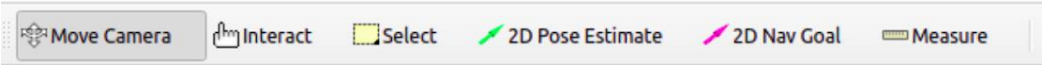
Figura 24 - Rviz



Fonte: autor

Para efeito de teste, foi utilizado o próprio Rviz para analisar a navegação autônoma. A Figura 25 apresenta duas ferramentas importantes para a aplicação, localizadas no canto superior esquerdo da janela do Rviz. Primeiramente é necessário informar a posição inicial estimada da cadeira por meio da tecla 2D *Pose Estimate*. Ao selecionar a posição, uma seta verde aparecerá, sendo possível indicar a orientação inicial estimada do robô. Depois desse passo, pode-se determinar o destino da cadeira, por meio da tecla 2D *Nav Goal*.

Figura 25 - Ferramentas Rviz

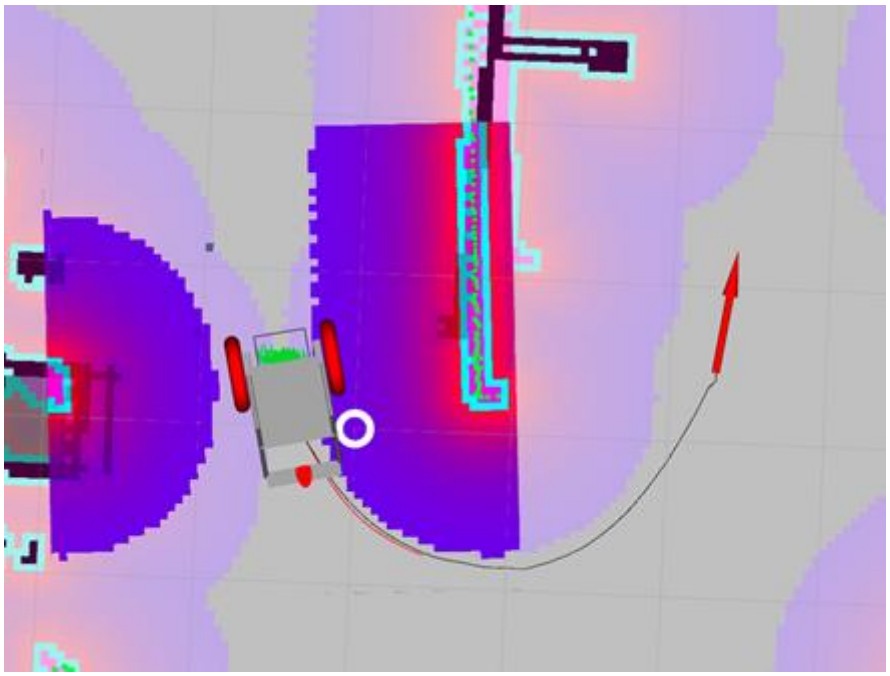


Fonte: autor

A Figura 26 apresenta a janela do Rviz durante a navegação autônoma da cadeira de rodas. O local onde a cadeira precisa alcançar está representada com uma seta vermelha, determinada pela tecla 2D *Nav Goal*. Nota-se que da cadeira saem duas linhas, sendo uma preta

e outra vermelha. A linha preta representa o global path, sendo que se inicia no robô e termina no objetivo. Já a linha vermelha representa o local path, sendo dinâmica ao decorrer do tempo, escolhendo a melhor trajetória de acordo com a leitura feita pelo LIDAR. Além da trajetória, o local *path planner* é representado pelas áreas coloridas ao redor da cadeira. É possível também visualizar o funcionamento do amcl por meio dos pontos verdes na parte posterior da cadeira, que representa as possíveis poses da cadeira de acordo com os algoritmos de Localização de Monte Carlos Adaptativa.

Figura 26 - Visão de movimentação da cadeira no Rviz

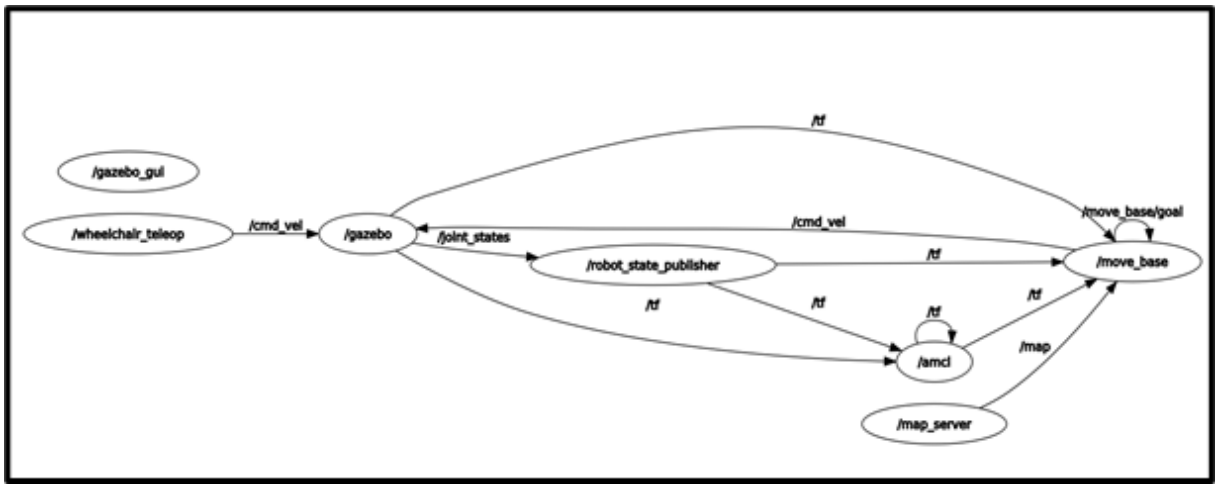


Fonte: autor

4.6. INTEGRAÇÃO ENTRE NÓS

É possível visualizar graficamente a integração dos nós do sistema através do plugin *rqt_graph*, conforme visto na Figura 27.

Figura 27 - Integração entre nós



Fonte: autor

O mapa estático é publicado pelo nó map_server no tópico map apenas uma vez, pois o mapa não muda, sendo nav_msgs/OccupancyGrid o tipo da mensagem. O nó move_base é subscrito por esse tópico, e recebe o nó para realizar o planejamento da navegação (QUIGLEY ET AL., 2015; ROS.ORG, 2020).

O nó robot_state_publisher publica no tópico tf a pose 3D dos links do robô, recebendo como entrada o modelo URDF através do parâmetro robot_description e a posição das juntas do tópico joint_states é publicado pelo nó gazebo (QUIGLEY ET AL., 2015).

O nó gazebo subscreve o tópico cmd_vel referente aos comandos de controle da velocidade da cadeira de rodas, sendo a mensagem do tipo geometry_msgs/Twist. O sistema contém dois nós que publicam nesse tópico: move_base e wheelchair_teleop. O move_base está relacionado com o movimento autônomo da cadeira, enquanto o wheelchair_teleop é responsável pelo comando manual através das setas do aplicativo.

O recorrente tópico tf registra a variação das relações entre os frames de coordenada, sejam eles pertencentes ao robô ou ao mapa. Esse histórico é utilizado como referência para quaisquer processos que utilizam informações relativas à posição. (OLIVEIRA ET AL., 2019)

4.7. ÁRVORE DE TRANSFORMAÇÕES

O Pacote TF fornece uma ferramenta chamada *view_frames* que possibilita uma análise gráfica da árvore de transformações atual. Para utilizar tal ferramenta, basta digitar o comando da Figura 28 no terminal Linux, então é criado um arquivo *pdf* com a cadeia de dependência de transformações.

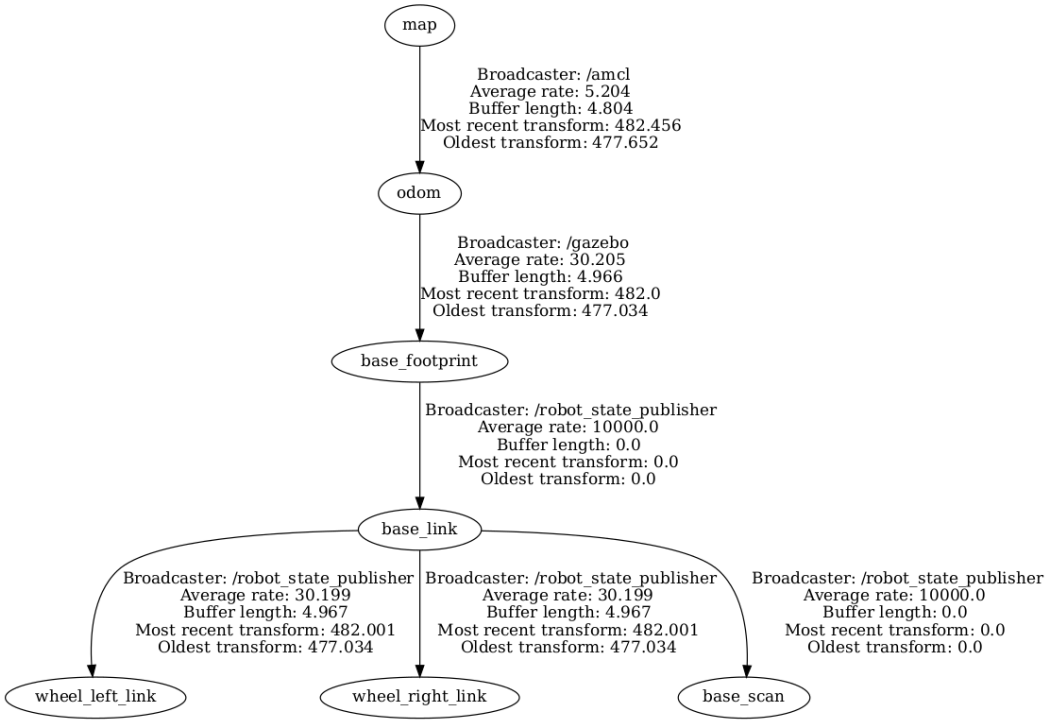
Figura 28 - Comando view frames

```
rosrun tf2_tools view_frames.py
```

Fonte: autor

A Figura 29 apresenta a árvore de transformações da cadeira de rodas. Podemos ver que o nó *gazebo* é o responsável pela transformação dos *frames odom* e *base_footprint*, o nó *amcl* dos frames map e odom, enquanto as outras transformadas são descritas pelo nó *robot_state_publisher*. O nó *robot_state_publisher* publica no tópico tf a pose 3D dos *links* do robô, recebendo como entrada o modelo *URDF* através do parâmetro *robot_description* e a posição das juntas do tópico *joint_states* publicado pelo nó *gazebo*.

Figura 29 - Árvore de transformações da cadeira de rodas



Fonte: autor

4.8. DESENVOLVIMENTO APLICATIVO ANDROID

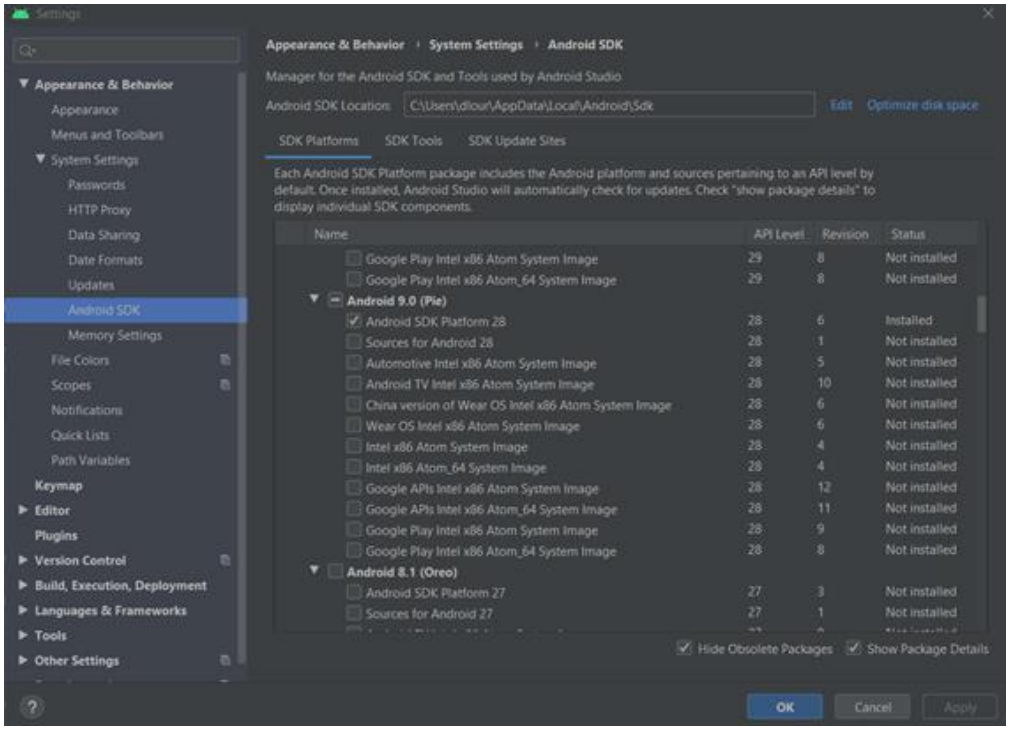
Para desenvolver o aplicativo Android utilizamos a *IDE Android Studio* e a linguagem de programação Java. Para iniciar o desenvolvimento do app configuramos alguns parâmetros para que a IDE funcione conforme planejado.

4.8.1. Configurações

No momento do desenvolvimento a última versão de Android lançada foi o Androird 11.0 (R), porém não foi utilizado esta versão, pois geralmente, as versões mais recentes contam com possíveis bugs que prejudicam o andamento do projeto, por isso foi escolhido uma versão um pouco mais antiga que já se mostrou muito estável, que foi a Android 9.0 (Pie).

Para instalar os pacotes necessários desta versão, é necessário acessar o *SDK Manager* do *Android Studio*, encontrados em: File > Settings > System Settings > Android SDK, conforme apresentado na Figura 30 para instalar a versão de Android desejada.

Figura 30 – Versão Android



Fonte: autor

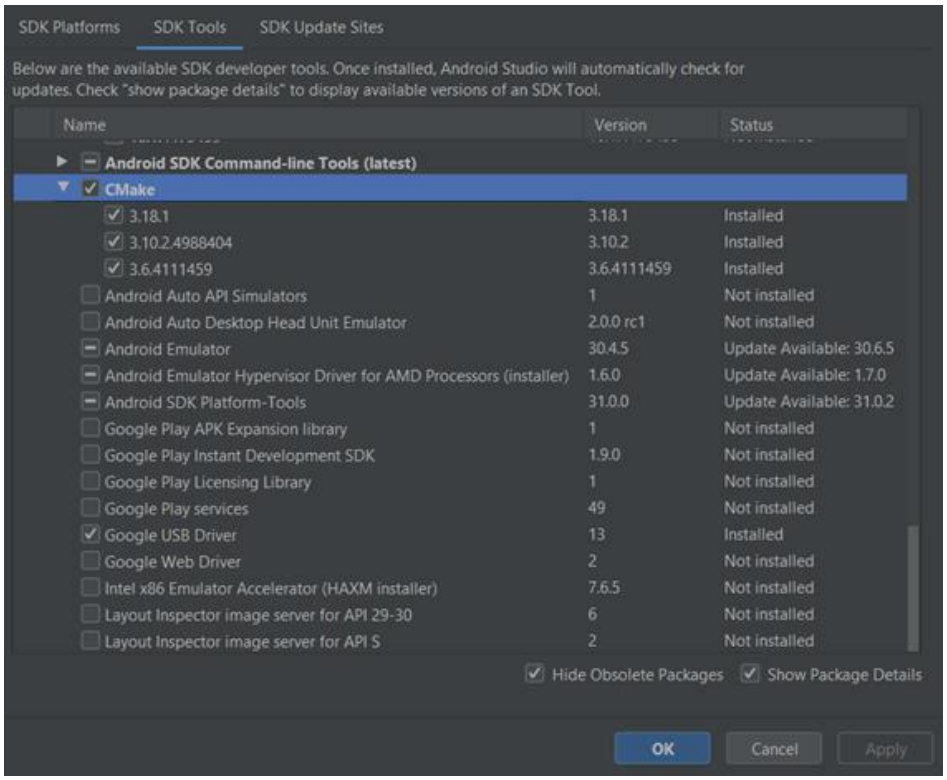
Ainda nesta tela do *SDK Manager*, é possível navegar entre as guias e desta vez serão feitas as configurações do *SDK Tools*. O primeiro item do guia é o *Android SDK Build Tool*, que irá auxiliar a compilar e testar o código. Foi instalado a versão 28.0.3 pois o SDK compatível com o Android 9.0 é o *Platform* 28, e este por sua vez é a versão mais atual deste SDK.

O próximo passo é instalar as ferramentas do NDK (Kit de desenvolvimento nativo) que fornece bibliotecas da plataforma para gerenciar atividades. Através de algumas pesquisas, foi identificado que o serial 21.1.6352462 se mostrou o mais ideal para seguir com o desenvolvimento por questões de compatibilidade.

Por último e não menos importante, a instalação das ferramentas, CMake (ferramenta de compilação externa para criar bibliotecas nativas) e Google *USB Driver* que se faz útil quando o aplicativo é testado em algum aparelho celular físico.

Na Figura 31 é demonstrada como fica a configuração após todos esses passos.

Figura 31 - Configução final



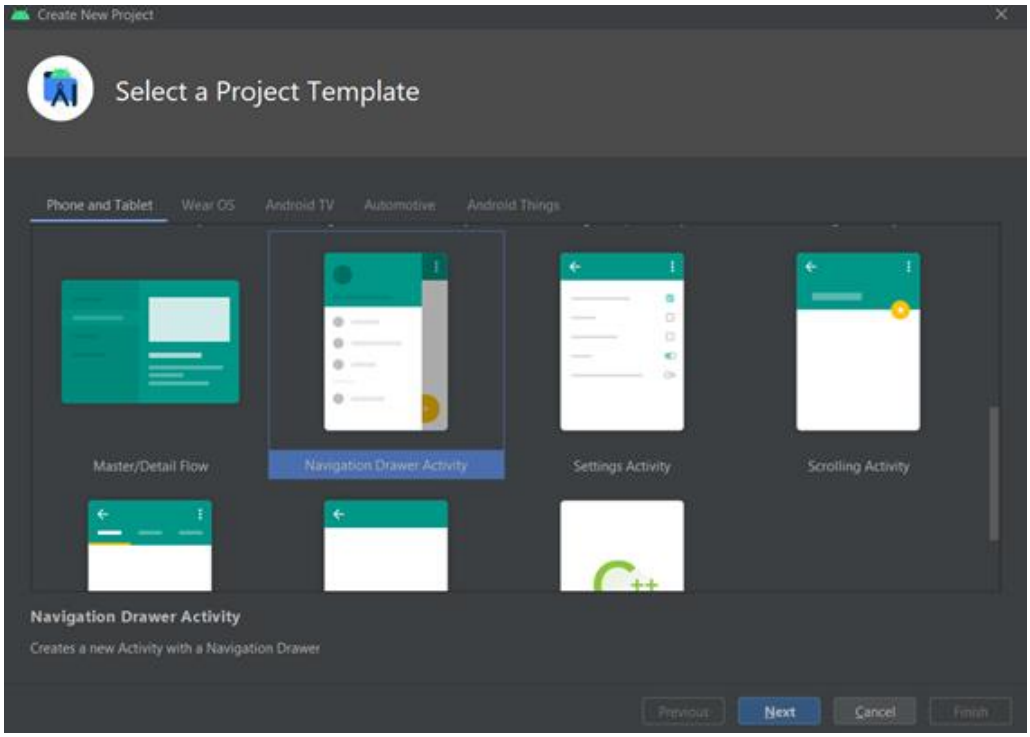
Fonte: autor

4.8.2. Criação do projeto

Após finalizar as configurações da IDE inicia-se o projeto, e o primeiro passo para isso é criá-lo no *Android Studio*. O primeiro passo para criação é selecionar o *Template* (modelo de aplicativo) que será utilizado.

O *Android Studio* possui uma gama variada de *Templates* para facilitar o desenvolvimento. Para o projeto foi utilizado o *Navigation Drawer Activity*, que é o modelo que possui a opção de navegação por abas laterais pelo aplicativo. Como demonstrado na Figura 32 a seleção do *Template*.

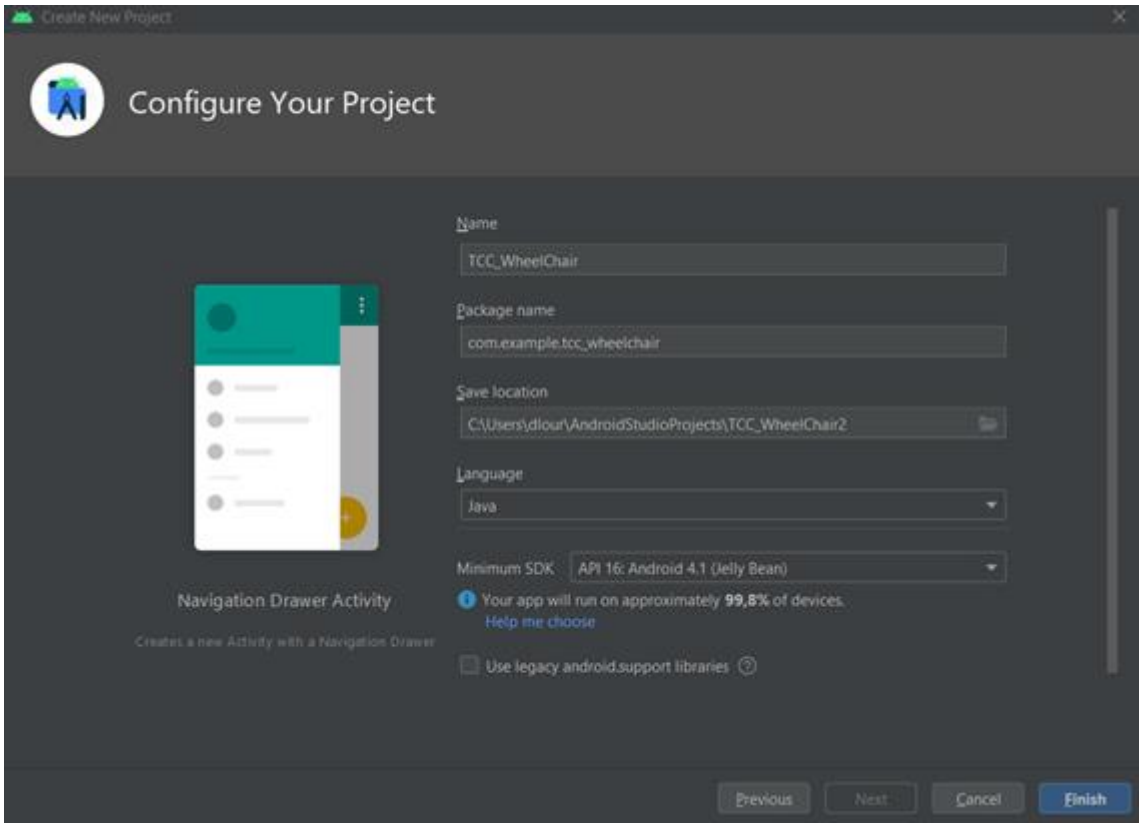
Figura 32 - Criação de um novo projeto



Fonte: autor

Após escolher o modelo da aplicação, nomeie o projeto baseado no tema proposto. Nesta fase também é selecionado a linguagem de programação a ser utilizada (no caso, Java) e o SDK mínimo que o aplicativo será compatível (versão de Android mínima para o funcionamento do mesmo), o qual utilizou o *Android 4.1 JellyBean* como exemplificado na Figura 33. Após todo esse processo inicial já pode trabalhar no desenvolvimento do mesmo.

Figura 33 - Configuração do projeto



Fonte: autor

4.9. PROGRAMAÇÃO

O escopo do projeto foi pensado para conter duas telas de navegação do Template Navigation Drawer Activity, uma para a movimentação manual da cadeira e outra para a autônoma.

Para desenvolver a primeira tela de movimentação manual, foram adicionados 5 ImageButtons (imagens representando botões com evento de toque) sendo 4 setas, uma cada direção que a cadeira possa se movimentar e um botão de STOP para suspender o movimento da cadeira. Para associar as imagens aos ImageButtons foi necessário adicionar as imagens em um repositório chamado drawable, para que o Android Studio adicione no aplicativo.

Após adicionar os botões, houve o dimensionamento do tamanho e a organização dos mesmos na tela. Importante citar os constraints (ferramenta para fixar os botões em um lugar específico da tela), pois, caso não houvesse, ao compilar o projeto as imagens iriam sobrescrever umas às outras no canto superior direito.

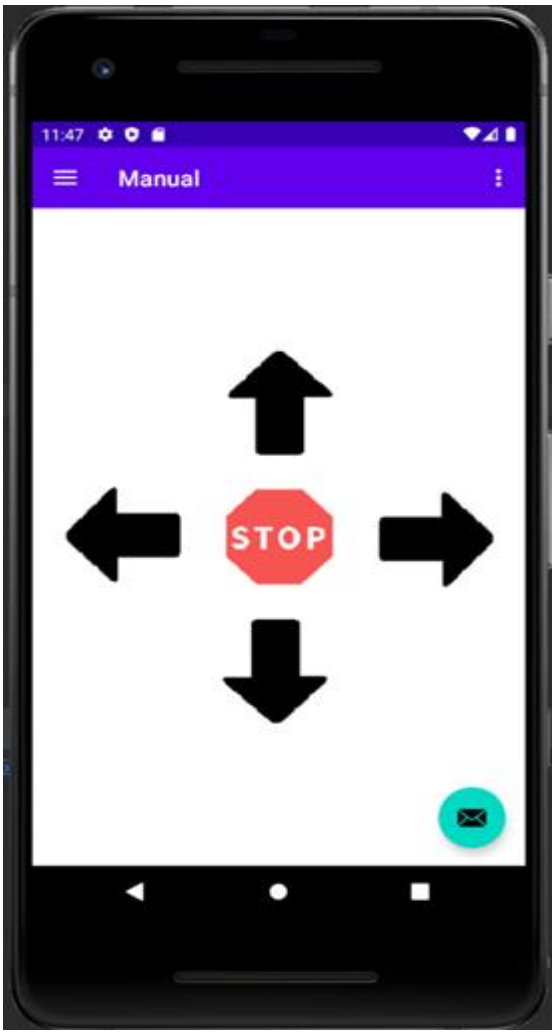
Para que esses botões exerçam alguma atividade no aplicativo, foram criados métodos que vão ditar o que cada botão irá fazer. A funcionalidade destes métodos é relativamente simples, ao pressionar o botão o aplicativo criará uma comunicação socket com o

wheelchair_teleop para transmitir qual comando será executado por ele, dependendo de qual seta ou botão será pressionado. Em resumo, cada ImageButton possui o seu método atrelado ao evento de clique para enviar comandos à cadeira.

Para abrir a comunicação através do socket foi criado uma classe específica a qual chamamos de CommSocket.java. Esta classe por sua vez possui um método que cria uma thread que será executada ao fundo ao ser chamada. Esta thread faz a instância de um novo socket (através da classe importada ao código Java.net.Socket), tendo como parâmetro o IP e a porta onde será feita a comunicação das aplicações. Além de instanciar o socket, esta thread também possui a função de enviar e receber informações, ou seja, é nela que ocorre o tráfego de dados entre as duas aplicações.

Na Figura 34 pode-se visualizar a aparência da primeira tela de navegação:

Figura 34 - Aplicativo módulo por setas

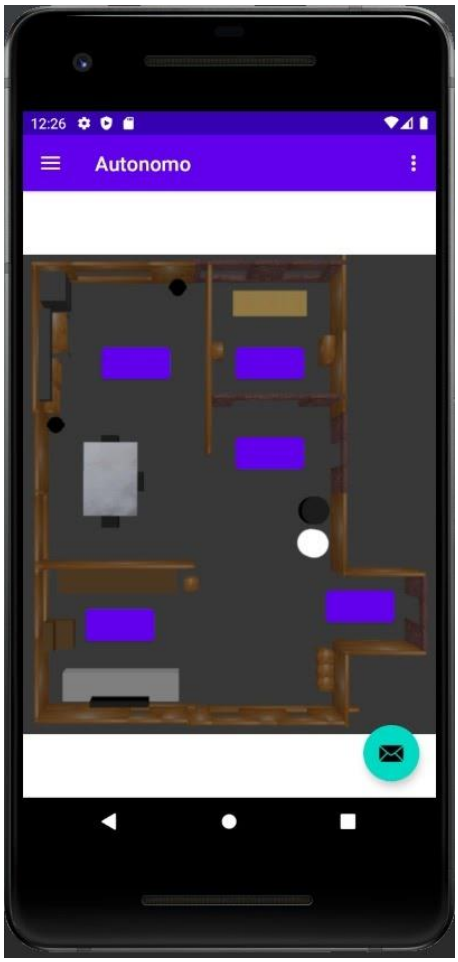


Fonte: autor

A segunda tela de navegação como demonstrado na Figura 35, possui o funcionamento

similar à primeira, porém ao invés de *ImageButtons*, apresenta a planta da casa da simulação que foi adicionada através de um *ImageView* (ferramenta para adicionar imagens ao aplicativo com a finalidade ilustrativa, sem interações) e em cada cômodo da casa há botões que serão pressionados para que a cadeira se desloque até lá.

Figura 35 - Aplicativo módulo autônomo



Fonte: autor

Assim como na primeira tela, os botões possuem métodos próprios para serem executados, ao receber um evento de clique. Estes métodos também iniciam uma comunicação por *socket*, este é com um *Shell Script* que é executado pelo robô na simulação. O *socket* também faz parte da uma *thread* que é executada ao fundo para estabelecer a comunicação. Os dados enviados são coordenadas estabelecidas para cada cômodo da casa, fazendo com que a cadeira se desloque até o destino.

O *Shell Script* citado foi desenvolvido para fazer a comunicação por *socket* com o aplicativo e executar o comando *rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped '{header: {stamp: now, frame_id: "map"}, pose: {position: {x: ,*

y: , z: }, orientation: {w: 1.0}}}', onde x, y e z serão as coordenadas passadas pelo aplicativo em referência ao cômodo desejado. Este comando faz com que a cadeira se mova de forma autônoma até a coordenada que foi enviada, desviando de possíveis obstáculos.

4.10. HARDWARE

Devido às dificuldades de encontros presenciais no período de desenvolvimento do projeto, este foi implementado apenas no ambiente virtual, porém a idealização caso este fosse passado para o meio físico.

O hardware deste projeto é relativamente simples, contando com uma cadeira manual, onde será adaptado os eixos dos motores diretamente nas rodas da cadeira. Foi escolhido este modelo pelo leve peso que se apresenta, pois com a adição de motores, baterias, LIDAR e placas de circuito terá um acréscimo de 20 a 30 kg, a bateria e as placas serão adaptadas com suportes abaixo do assento, já o LIDAR seria acoplado entre os pés da cadeira.

4.10.1. LIDAR

O LIDAR (*Light Detection And Ranging*) é um sistema de sensoriamento remoto ativo que é utilizado como sensor para realizar o mapeamento, determina a localização e detecta obstáculos. O princípio de funcionamento se baseia na emissão de luz para realizar a medição de distância. Ao atingir o ambiente ou um obstáculo, a luz é refletida, retornando ao sensor, onde é registrada. A distância então é calculada de acordo com o tempo de retorno (LIDAR, 2021).

O modelo escolhido para compor o projeto foi o LDS-01, que realiza uma varredura omnidirecional de 360 graus. Ele é apresentado na Figura 36. Alguns parâmetros importantes do modelo são:

- Distância de alcance de 0,12 a 3 metros;
- Tensão de alimentação de 5VCC (±5%);

Figura 36 - LDS-01

Fonte: Robotis, 2021

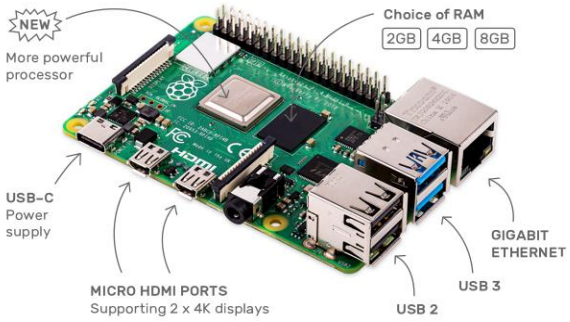
4.10.2. Raspberry

Na Figura 37 apresentamos o Raspberry, um minicomputador que deve receber os sinais advindos do LIDAR e do aplicativo no celular para realizar o processamento no ROS e gerar os sinais de comando para o Arduino.

O Raspberry escolhido foi o Pi 4 Model B, modelo mais atual durante o desenvolvimento da lista de materiais. Alguns parâmetros importantes do modelo são:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz;
- 8GB LPDDR4-3200 SDRAM;
- 2.4 GHz e 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE;
- Tensão de alimentação de 5VCC via conector USB-C.

Figura 37 - Raspberry Pi 4 Model B



Fonte: Raspberry PI, 2021

4.10.3. Arduino

O microcontrolador escolhido foi o Arduino Mega 2560 REV3 que está apresentado na Figura 38, este receberá sinais de controle do Raspberry e enviará comandos para os motores.

Alguns parâmetros importantes do modelo são:

- Oscilador de cristal de 16 MHz;
- Corrente DC por Pino de I/O de 20 mA
- Memória flash 256 KB, SRAM 8 KB;
- Tensão de 5VCC.

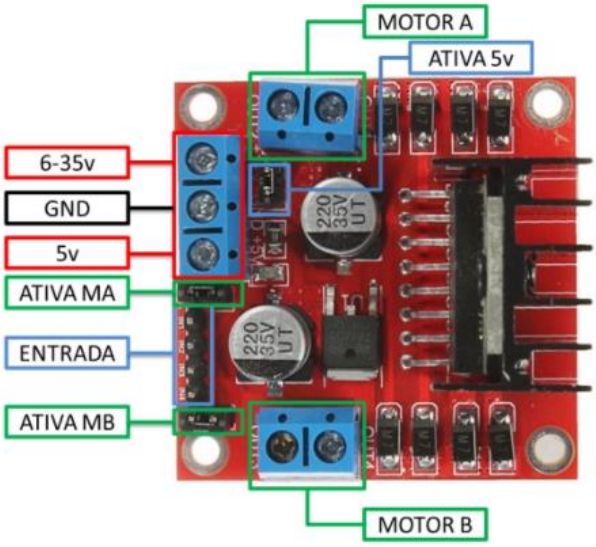
Figura 38 - Arduino MEGA 2560

Fonte: Arduino.cc, 2021

4.10.4. Placas de alimentação

Para o controle de velocidade dos motores foi pesquisado e determinado que a Ponte H L298n que está apresentado na Figura 39 que é um módulo externo para Arduino, atenderia a necessidade, a sua finalidade é o controle de velocidade deles, pelo fato de ser um módulo periférico do Arduino apresenta uma fácil utilização.

Figura 39 - Ponte H L298n



Fonte: FilipeFlop, 2013

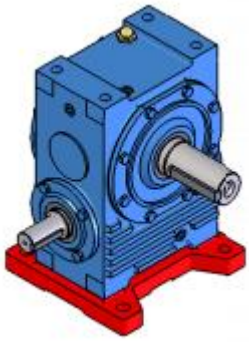
4.10.5. Motores e redutores

Para a montagem dos motores da cadeira, foi pesquisado e escolhido dois motores de corrente contínua um para cada roda traseira, os modelos são MBT86Vl da Leroy Somer, como demonstrado a Figura 40, são motores de aplicação industrial como esteiras alimentadoras, eixos de fresas, movimentações de cargas entre outras aplicações. A tensão nominal é de 30V, trabalhando com uma variação de tensão de operação de 24V a 48V, e rotação de 1300 rpm com um torque de 2N.m(20,4kgf.cm); por esses valores de torque e rotação não serem o que precisamos serão usados redutores, considerando para movimentar uma carga estimada de 100 à 150 kg (pessoa mais cadeira) e uma rotação de 65 rpm calculamos que serão necessárias uma caixa de redução 1:20, CR-200 em cada motor semelhante a Figura 41.

Figura 40 - Motor MBT86Vl, Leroy Somer

Fonte: Robocore, 2021

Figura 41 – Redutor - CR-200

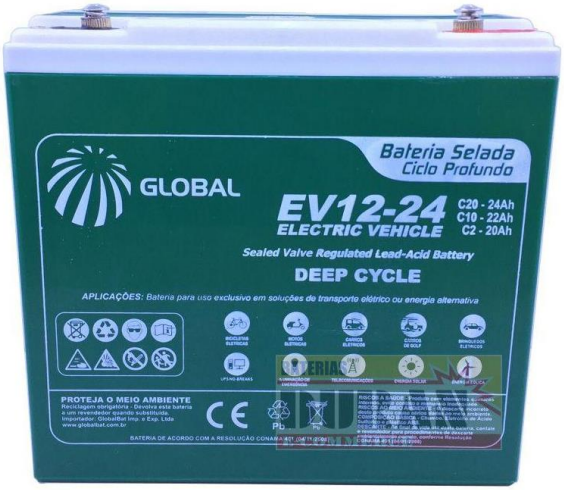


Fonte: Cesalto, 2021

4.10.6. Bateria

As baterias escolhidas para alimentação tanto dos motores como das placas de comando estão demonstradas na Figura 42, seriam duas baterias de 12V, baterias próprias para cadeiras de rodas da marca Global.

Figura 42 - Bateria 12V



Fonte: Global Power, 2020

4.10.7. Regulador de Tensão

Como o sistema inteiro é alimentado pelas mesmas baterias há a necessidade de reduzir a tensão para a alimentação dos módulos, Arduino e Raspberry. Para garantir a compatibilidade de tensão será utilizado módulos reguladores de tensão LM2576HV, módulos conforme Figura 43, com saída de 5V e corrente máxima 3A.

Figura 43 - LM2576HV

Fonte: Eletrodex, 2021

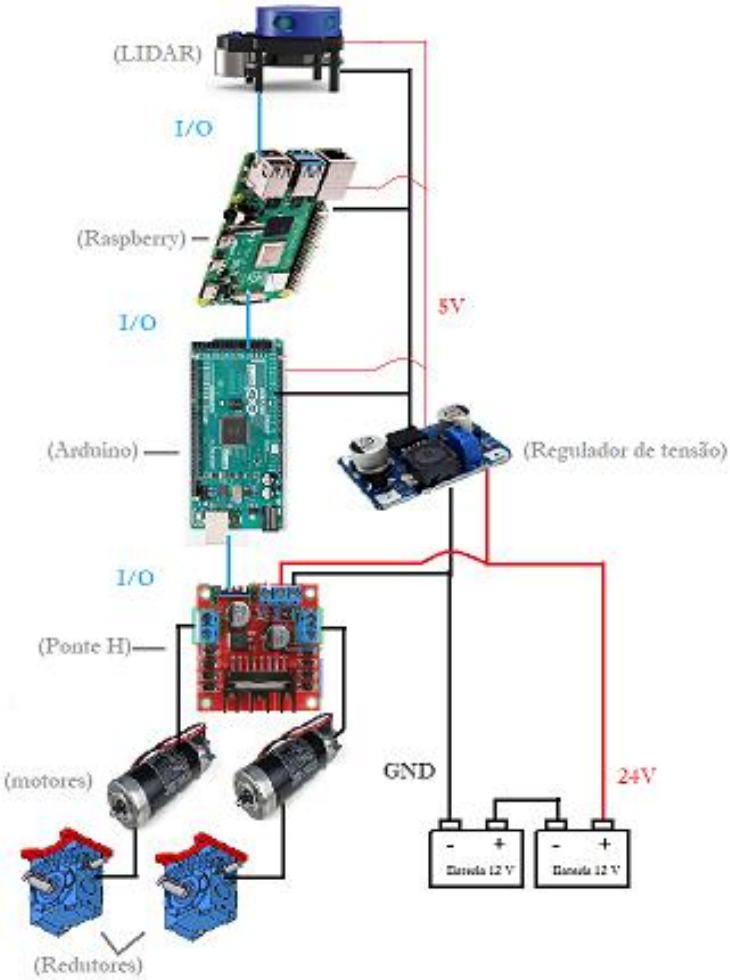
4.10.8. Circuitos e Diagramas

Com as duas baterias fornecendo 24V temos a alimentação básica da cadeira,

alimentando a ponte H que distribuirá a tensão para os motores e o regulador para fornecer uma tensão de 5V para o LIDAR, Raspberry.

Com o Raspberry recebendo as informações vindas do aplicativo e do LIDAR, ele processa no ROS e envia o comando para o arduino que por sua vez envia a informação de acionamento e velocidades dos motores para a ponte H, conforme Figura 44.

Figura 44 - Esquema de ligação entre partes do Hardware



Fonte: autor

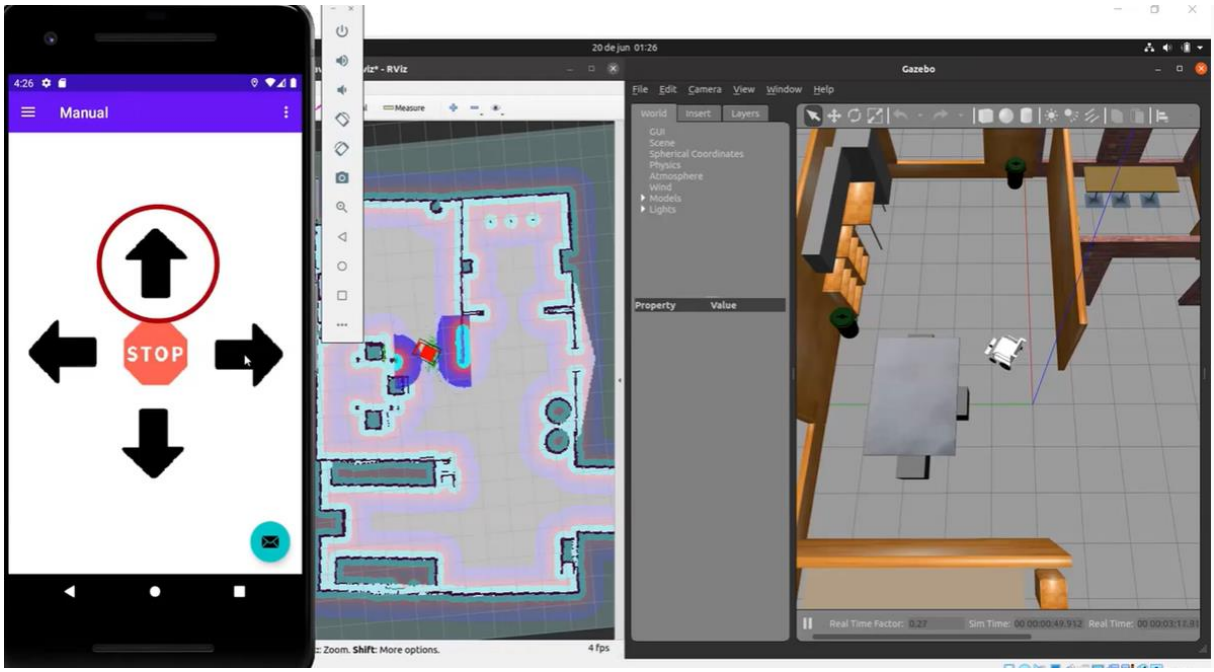
5. RESULTADO

O objetivo do trabalho foi construir uma cadeira de rodas com duas formas de controle que pudesse proporcionar ao usuário uma autonomia com facilidade de manuseio, o objetivo foi alcançado onde os resultados estão apresentados abaixo:

Na Figura 45 há três visões necessárias para verificar o funcionamento do projeto o primeiro é do *Android Studio* que simula o aplicativo no celular, ao lado dele, o Rviz que mostra a visão da cadeira através do LIDAR, o principal responsável por mapear o local, e por último o Gazebo que representa o comportamento real da cadeira conforme as escolhas de comando no celular.

Ao tocar na seta de cima do aplicativo, a cadeira se movimenta para frente e se ação se repetir mais uma vez a velocidade aumenta chegando até a 0,3m/s, se pressionadas as setas da esquerda ou direita a cadeira incrementa uma velocidade angular, desta forma virando e pode chegar a uma velocidade de 2,84rad/s.

Figura 45 - Simulação do sistema completo, controle por setas

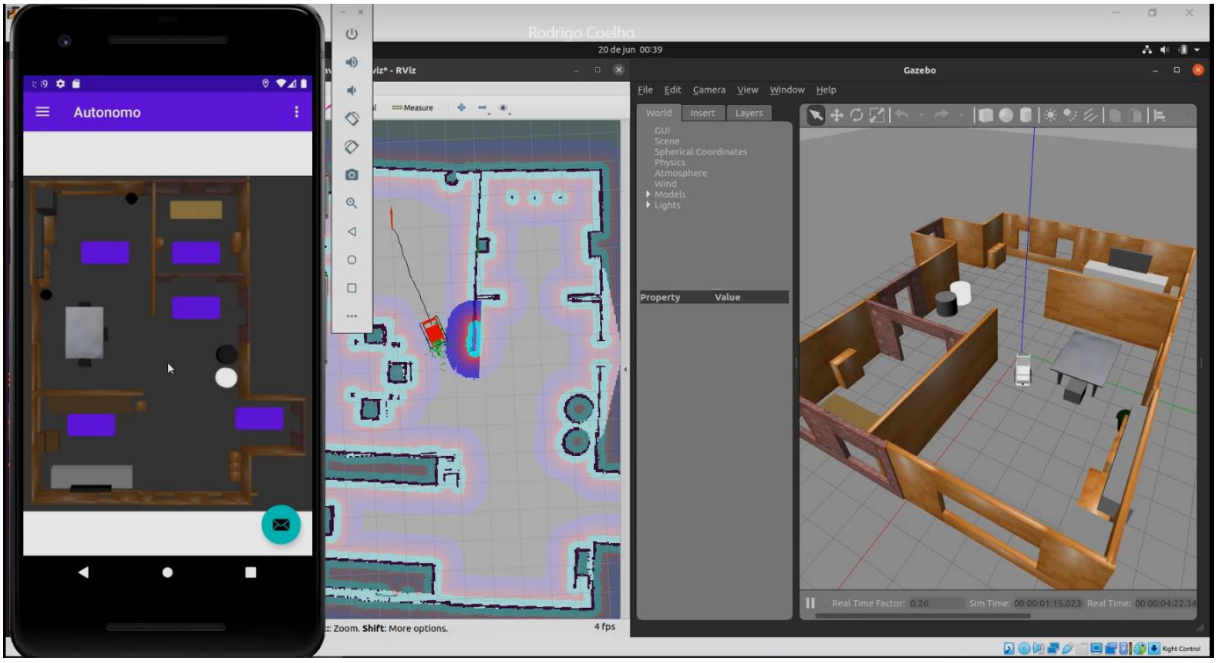


Fonte: autor

A segunda forma de controle, o modo autônomo de navegação, possui locais de seleção pré-definidos de acordo com a planta mostrada no aplicativo que está representado na Figura 46, tais locais podem ser acionados ao pressionar os botões em roxo e onde o usuário tocar, a cadeira recebe as coordenadas como demonstrado no Rviz representado por uma linha e o

caminho que a cadeira faz por uma seta vermelha.

Figura 46 – Simulação do sistema completo, controle por locais pré-determinados

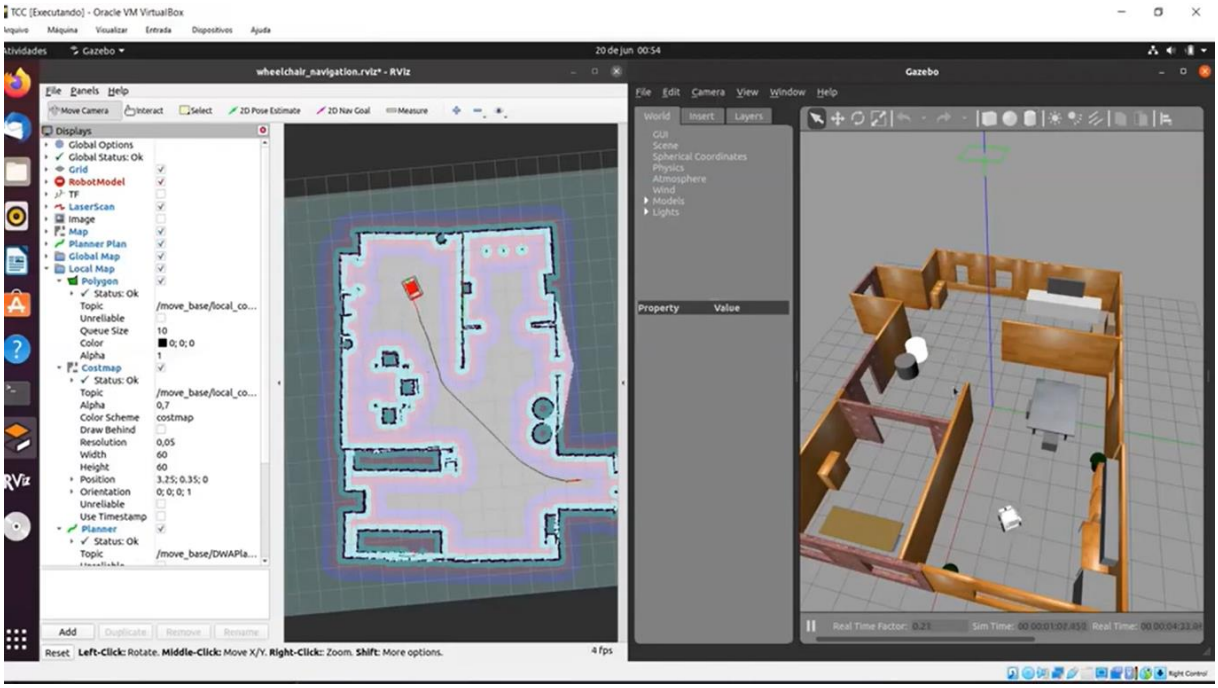


Fonte: autor

Ao escolher outro local a cadeira recebe uma nova coordenada, traça a rota e começa o cominho até o ponto conforme a Figura 47, se houver obstáculos como demonstrado na Figura 48 o LIDAR verifica tais obstáculos e recalcula as rotas para desviar. Uma coisa importante é o fato de que a cadeira de forma autônoma não anda de costas, quando é selecionado um destino a cadeira vira e se move até o local.

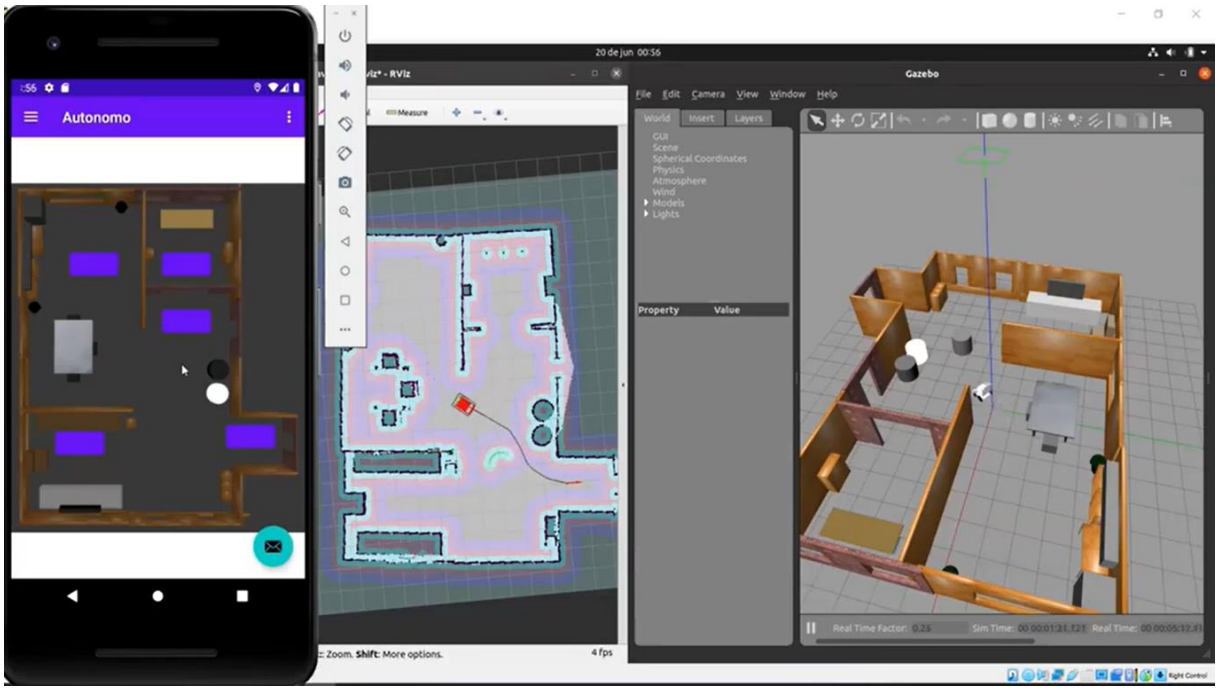
Desta forma a cadeira cumpriu seus objetivos, funcionando de forma adequada tanto em seu modo por setas, como principalmente em seu modo autônomo, indo de para os locais selecionados de forma correta, desviando de obstáculos e alterando rotas em caso de engano.

Figura 47 - Simulação completa, segunda escolha de local.



Fonte: autor

Figura 48 - Detecção de Obstaculos



Fonte: autor

6. CONCLUSÕES

Escolhemos este projeto pois o desenvolvimento de tecnologias assistivas são de extrema importância, ganhando cada vez mais visibilidade com a evolução da robótica. Apesar de existir diversos tipos de cadeiras no mercado, buscamos apresentar uma solução de baixo custo que abrangesse um maior público com algum tipo de dificuldade de locomoção.

O projeto foi idealizado para ter um protótipo construído, porém em decorrência da situação pandêmica global e agravamento no país, optamos em trabalhar de forma remota com o intuito de resguardar a saúde e bem estar dos integrantes e desta forma definimos desenvolver o projeto em simulação, buscando representar de forma muito próxima do real o comportamento da cadeira.

Para a elaboração deste projeto adquirimos o conhecimento em diversos assuntos como o Sistema ROS, simuladores, conceitos teóricos de navegação, desenvolvimento android entre outros já citados, deparamos com alguns dificuldades no decorrer do projeto como por exemplo a conexão entre diferentes linguagens de programação, pois desenvolvemos os códigos do ROS na linguagem Python e no android utilizamos a linguagem Java. Outra dificuldade foi em adquirir o arquivo URDF, que é o descritivo do robô para a simulação. Houve mudanças ao longo do semestre em relação à ideia inicial apresentada, pois iríamos utilizar o simulador V-REP mas o Gazebo se mostrou eficiente e com fácil integração com ROS.

Com a conclusão do projeto, observamos que ainda há possibilidade de expansão das ideias elaboradas, assim como, a construção de um protótipo físico, entretanto com as inovações trazidas através dos sistemas desenvolvidos mostramos uma nova forma de utilizar os sistemas de localização de robos e integração entre tecnologias distintas como foi mostrado ao utilizar o sistema android em conjunto com ROS, para realizar a interface homem máquina e desta forma tornar a tecnologia autonavegação mais acessivel e com mais opções de uso para que atenda o objetivo de proporcionar conforto, acessibilidade e autonomia a usuários que podem vir a fazer uso da cadeira de rodas autônoma.

REFERÊNCIAS

ARDUINO.CC. Store. Disponível em: https://store.arduino.cc/usa/mega-2560-r3. 2021. Acesso: 08 jun. 2021.

CANALTECH. Introdução ao shell Script. 2014. Disponível em: https://canaltech.com.br/linux/Introducao-ao-Shell-Script/ Acesso em: 08 de jun. 2021.

CESTALTO. [20--]. Disponível em: http://cestalto.com.br/novo/PDF/LINHA-CR/Conj-Completo-CR-200.pdf. Acesso: 08 jun. 2021.

DEVELOPERS. Plataforma. 2020. Disponível em: https://developer.android.com/guide/platform?hl=pt-br. Acesso: 08 jun. 2021.

DEVMEDIA. Introdução ao Shell Script no Linux. 2015. Disponível em: https://www.devmedia.com.br/introducao-ao-shell-script-no-linux/25778. Acesso: 08 de jun. 2021.

DEVMEDIA. Tutorial de Android Studio. 2016. Disponível em: https://www.devmedia.com.br/tutorial-de-android-studio/34003. Acesso: 08 jun. 2021.

ELETRODEX. Módulo LM2576HV Step Down 3A Buck 5 a 50 VDC. [20--]. Disponível em: https://www.eletrodex.com.br/modulo-lm2576hv-step-down-3a-buck-5-a-50-vdc.html. Acesso: 08 jun. 2021.

FILIPEFLOP. Motor DC com Driver Ponte H L298N. 2013. Disponível em: https://www.filipeflop.com/blog/motor-dc-arduino-ponte-h-l298n. Acesso: 08 jun. 2021.

GLOBAL POWER. Baterias. 2020. Disponível em: https://globalpower.com.br/produtos/baterias/. Acesso: 08 jun. 2021.

GREWAL, H. et al. LIDAR-Based Autonomous Wheelchair. IEEE Sensors Applications Symposium (SAS): s.n., 2017.

IBGE, População residente por tipo de deficiência permanente. 2010. Disponível em: https://www.ibge.gov.br/estatisticas/sociais/populacao/9662-censo-demografico-2010.html?edicao=9749&t=destaques. Acesso em: 08 jun. 2021.

JOSEPH, Robot Operating System (ROS) for Absolute Beginners: Robotics Programming Made Easy. p. 151, s.n., Apress, 2018.

KOUBAA, ROBOT OPERATING SYSTEM (ROS): The Complete Reference (Volume 1), p. 137, s.n., Springer, 2016.

LIDAR, Princípio de funcionamento de um LIDAR aéreo. 2021. Disponível em: https://www.lidar.com.br/index.php/tecnologia/principio-de-funcionamento-de-um-lidar-aereo. Acesso em: 08 jun. 2021.

MACHADO, Julio. Programação orientada a objetos com Java. PUC, Rio Grande do Sul, 2018. Disponível: https://www.inf.pucrs.br/~danielc/cursos/pqts/java/Java1.pdf. Acesso em: 14 nov. 2020.

MORIN, D. Introduction to classical mechanics: with problems and solutions. Harvard University, Massachusetts: Cambridge University Press, 2008. p. 320. 2010

MURARKA, Aniket et al. Towards a Safe, Low-Cost, Intelligent Wheelchair. In: St. Louis, Missouri, Workshop On Planning, Perception And Navigation For Intelligent Vehicles: s.n., 2009.

OLIVEIRA, et al. Cadeira de rodas com deslocamento adaptável ao ambiente. Centro universitário da FEI: São Bernardo do Campo, 2019.

PANTUZA. O que são e como funcionam os sockets. 2017. Disponivel em: https://blog.pantuza.com/artigos/o-que-sao-e-como-funcionam-os-sockets. Acesso: 08 jun. 2021.

QUIGLEY, et al. PROGRAMMING ROBOTS WITH ROS: A Practical Introduction to the Robot Operating System, p. 3, s.n., O'Reilly Media, Inc., 2015.

RASPBERRY PI. Raspberry Pi 4. 2021. Disponível em: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/?resellerType=home. Acesso: 08 jun. 2021.

RETHINKROBOTICS. Rviz 2015. Disponível em: https://sdk.rethinkrobotics.com/wiki/Rviz Acesso: 08 jun. 2021.

ROBOTIS. LDS-01. [20??]. Disponível em: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/. Acesso: 08 jun. 2021.

ROS.ORG. AMCL. 2020. Disponível em: http://wiki.ros.org/amcl?distro=noetic. Acesso em: 08 jun. 2021.

ROS.ORG. MAP_SERVER. 2020. Disponível em: http://wiki.ros.org/map_server. Acesso em: 08 jun. 2021.

ROS.ORG. ROSLAUNCH. 2019. Disponível em: http://wiki.ros.org/roslaunch. Acesso em: 08 jun. 2021.

ROS.ORG. URDF. 2019. Disponível em: http://wiki.ros.org/urdf. Acesso em: 08 jun. 2021.

ROBOCORE. Motor Industrial 30V 1300RPM 80mm. 2021. Disponível em: https://www.robocore.net/motor-motoredutor/motor-leroy-somer-mbt86vl. Acesso: 08 jun. 2021.

SOLA, Joan. Simulataneous localization and mapping with the extended kalman filter, The Institut de Robòtica i Informàtica Industrial, 2014

UNIVERSIDADE JAVA. Java - Hello World. 2011. Disponível em: http://www.universidadejava.com.br/materiais/java-helloworld/. Acesso: 08 jun. 2021.

WHILL, Model Ci. 2020. Disponível em: https://whill.inc/us/model-ci. Acesso em: 14 nov. 2020.

YOOK, et al. Smart wheelchair using android smartphone for physically disabled people. Departamento de Tecnologia de Engenharia Elétrica, Faculdade de Tecnologia de Engenharia, Universidade Teknikal Malásia Melaka, Hang Tuah Jaya, 76100, Durian Tunggal, Malaca, Malásia. 2018.