Universidade Federal do Rio de Janeiro
Escola Politécnica

INTERFACE WEB PARA O ROBÔ MÓVEL COM ESTEIRAS UTILIZANDO ROS

Rôb Klér Soares da Silva Júnior

Projeto de Graduação apresentado ao Corpo Docente do Curso de Engenharia de Controle e Automação da Escola Politécnica da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro de Controle e Automação.

Orientador: Fernando Cesar Lizarralde

Rio de Janeiro
Maio de 2018

INTERFACE WEB PARA O ROBÔ MÓVEL COM ESTEIRAS UTILIZANDO ROS

Rôb Klér Soares da Silva Júnior

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

Examinado por:

Prof. Fernando Cesar Lizarralde, D.Sc.

Prof. Ramon Romankevicius Costa, D.Sc.

Eng. Alex Fernandes Neves, M.Sc

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2018

Soares da Silva Júnior, Rôb Klér

Interface Web Para o robô móvel com esteiras utilizando ROS / Rôb Klér Soares da Silva Júnior. – Rio de Janeiro: UFRJ/Escola Politécnica, 2018.

XI, 53 p.: il.; 29, 7cm.

Orientador: Fernando Cesar Lizarralde

Projeto de Graduação – UFRJ/Escola Politécnica/ Curso de Engenharia de Controle e Automação, 2018.

Referências Bibliográficas: p. 50 – 51.

1. Interface. 2. Escada. 3. Subida. 4. Controle. 5. Web. 6. ROS. I. Lizarralde, Fernando Cesar. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Título.

Dedicatoria

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação

INTERFACE WEB PARA O ROBÔ MÓVEL COM ESTEIRAS UTILIZANDO ROS

Rôb Klér Soares da Silva Júnior

Maio/2018

Orientador: Fernando Cesar Lizarralde

Departamento: Engenharia de Controle e Automação

Este trabalho apresenta uma aplicação web para a teleoperação de um robô móvel com esteiras. A interface foi projetada para trazer facilidades e melhorias através de informações sobre vários aspectos do robô, como a velocidade e a posição de esteiras e braços.

O controle embarcado no robô móvel é baseado em ROS. Neste trabalho foi utilizado o pacote *Rosbridge* para acessar dinamicamente todos os recursos oferecidos pelo ROS. Com a biblioteca *roslibjs* é possível ler, escrever e aproveitar outros serviços utilizando JavaScript e objetos JSON, um formato leve para transferência de dados. Dessa forma, tornou-se viável a criação de uma interface web que pode ser usada em diferentes OS e arquiteturas.

Além de trazer informações para auxiliar a teleoperação do robô, a aplicação também proporciona comandos relacionados à subida autônoma de escadas, uma maneira simples de visualizar as posições do braços, os modos de controle que estão sendo utilizados e opções para ligar e desligar dispositivos acoplados ao robô pelas portas lógicas (led, laser e kinect).

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A utilização de interfaces web permite criar ambientes interativos de fácil manipulação e grande acessibilidade, trazendo novas possibilidades para o campo da robótica LEE (2012).

As interfaces web para robôs podem expandir o alcance e o impacto de tecnologias robóticas. Com uma estrutura adequada, a utilização dessas interfaces possibilita o controle dos robôs a partir de qualquer lugar e com acesso através de diversos dispositivos, como telefones e computadores, ainda que utilizem sistemas diferentes. Além disso, permite a criação de laboratórios para compartilhamento de experimentos e desenvolvimento.

O desenvolvimento de páginas web baseadas em HTML e *Javascript* se popularizou de tal forma que muitas pessoas possuem conhecimentos básicos sobre essas tecnologias e são capazes de desenvolver ferramentas através delas, fazendo com que não sejam de uso exclusivo de programadores mais experientes (ALEXANDER *et al.*, 2012). Um dos objetivos da implementação da interface web é tornar o trabalho mais simples e expandir a robótica para um novo público, por não ser necessário conhecimento aprofundado do sistema robótico.

Neste trabalho será apresentada uma interface web baseada em *JavaScript* desenvolvido para rodar em um ambiente de compilação separado e menos complexo que o utilizado pelo robô, com a intenção de não ficar limitado às suas ferramentas OSENTOSKI *et al.* (2011). Além de ser multi-plataforma, podendo rodar em qualquer sistema operacional que possua um navegador web.

1.1 Objetivos

O objetivo deste trabalho é apresentar uma interface web elaborada para a teleoperação do robô DIANE mostrado na figura 1.1. Estes operacionais possibilitam o controle do robô por vários dispositivos, de diferentes sistemas, através de uma

aplicação web que oferece opções de controle e informações a fim de facilitar a teleoperação.

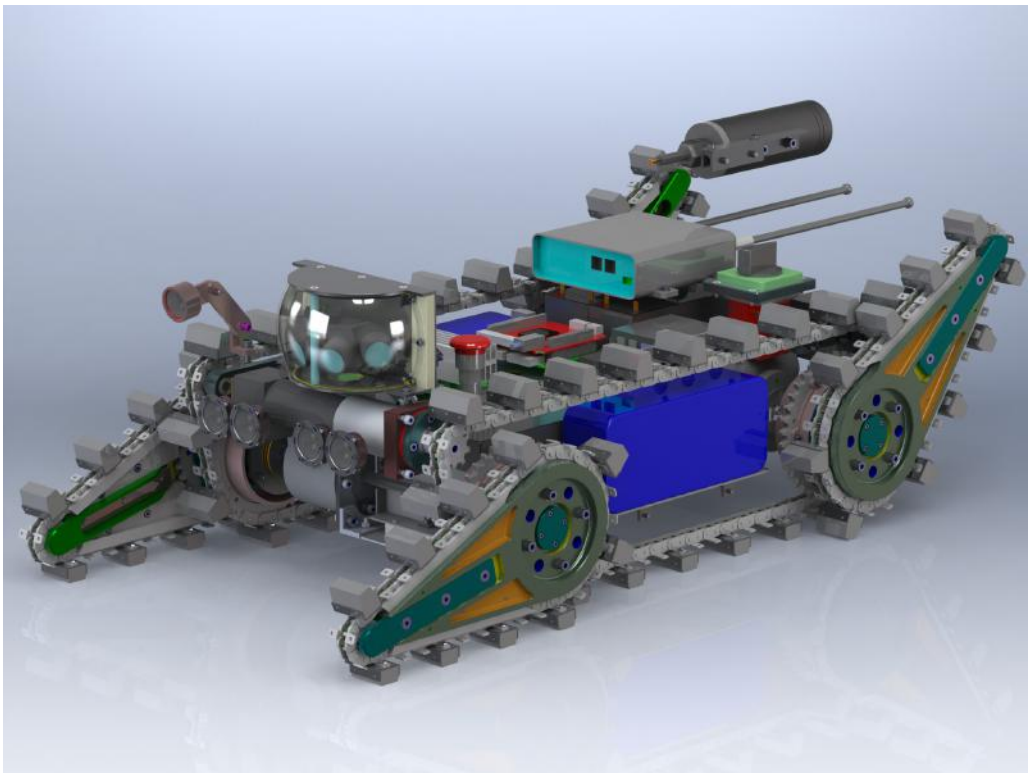

Figura 1.1: Projeto do robô DIANE.

A interface traz informações, possibilidades de controle, entre outras funcionalidades, sendo que algumas estão listadas abaixo.

- Controle por velocidade e posição de esteiras e braços;
- Informações físicas, tais como a temperatura do computador embarcado e a carga da bateria;
- Possibilidade de controle pelo teclado;
- Botões para ligar e desligar dispositivos ligados às portas lógicas;
- Velocidade e posição atual das esteiras e braços.

1.2 Revisão Bibliográfica

Muitas pesquisas em robótica fazem uso da Internet e de aplicações web para teleoperações remotas, promovendo o acesso a experimentos em robôs e coleta de dados de forma colaborativa TORIS *et al.* (2015).

Em CRICK *et al.* (2011) foi apresentado uma aplicação web no qual pessoas tentavam resolver um labirinto, mostrado na figura 1.2, o mais rápido que pudessem. Umas das maneiras de controle era através de uma câmera sobre o robô, a visão de abordo disponível ao usuário é mostrada na figura 1.3. A ideia desse experimento era obter um conjunto de dados que possibilitasse treinar o robô para resolver o labirinto de forma autônoma.

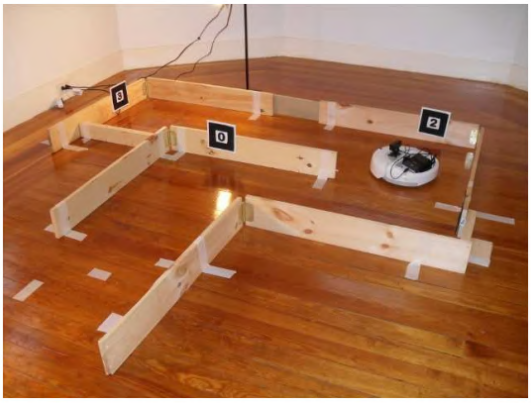

Figura 1.2: Labirinto criado para andar com o robô.

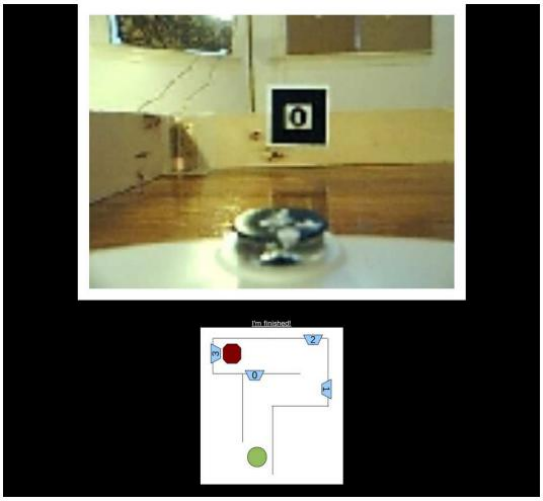

Figura 1.3: Visão obtida através da interface.

Em LEE (2012) foram desenvolvidas três aplicações web. Uma delas chamada *"PR2 Castle Builder"*, cujo objetivo é permitir a construção de castelos com blocos de encaixe e um PR2. Sua interface pode ser vista na figura 1.4. O *"PR2 commander"* tem como objetivo executar tarefas de forma autônoma, sendo seu comando passado por voz. Sua interface pode ser vista na figura 1.5. A última aplicação, chamada de *"PR2 Turntable Manipulator"* , é um robô interagindo com determinados objetos em uma mesa giratória, sendo as tarefas de apanhar e devolver o objeto à mesa decisões de quem está controlando o robô pela interface, a qual pode ser vista na figura 1.6.

Em CHUNG *et al.* (2014) desenvolveram uma página na qual pessoas criam modelos 2D de objetos, indicados pelo site, e o robô usa esses modelos para aprender e criar o seu próprio. A criação desta interface fez com que inúmeras pessoas pudessem contribuir para o crescimento da base de dados utilizada pelo robô. Um exemplo da página é mostrada na figura 1.7.

Em LOPEZ *et al.* (2009) desenvolveram uma página que permitisse aos estudantes o acesso remoto a um computador, no qual pudessem elaborar o código necessário e o testavam no robô. A proposta era de que os estudantes formassem grupos de duas a três pessoas e resolvessem o problema da *Hanoi Towers*, mostrado na figura

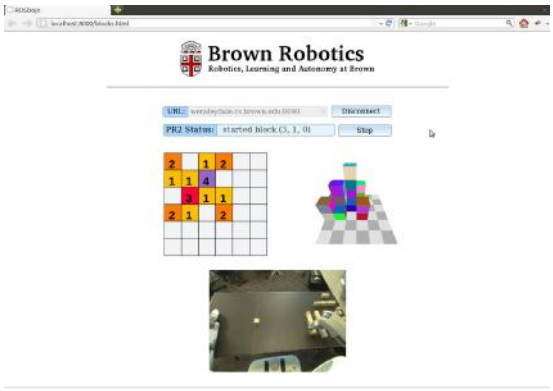

Figura 1.4: Página do *PR2 Castle Builder*.

Figura 1.5: Página do *PR2 Commander*.

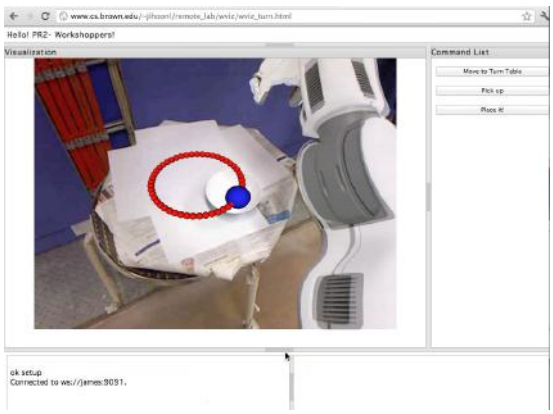

Figura 1.6: Página do *PR2 Turntable Manipulator*.

1.8. Para solucionar, era preciso que o robô movesse todas as peças da posição 1 para a 3, sendo uma por vez e nunca existindo uma peça sobre outra menor do que ela. Todos os estudantes trabalhariam no mesmo terminal de forma colaborativa, sendo possível a todos que editassem, compilassem e rodassem o código.

1.3 Robot Operating System - ROS

O Robot Operating System(ROS) foi o *framework* escolhido na atualização do software do robô DIANE pela sua capacidade de resolver inúmeros problemas que aparecem durante o desenvolvimento de aplicações para robôs como a comunicação entre os componentes internos e externos, possibilitando a realização de várias tarefas ao mesmo tempo QUIGLEY *et al.* (2009), e a facilidade para construir sistemas e aplicações sem a necessidade de re-escrever funcionalidades padrões TORIS *et al.* (2015).

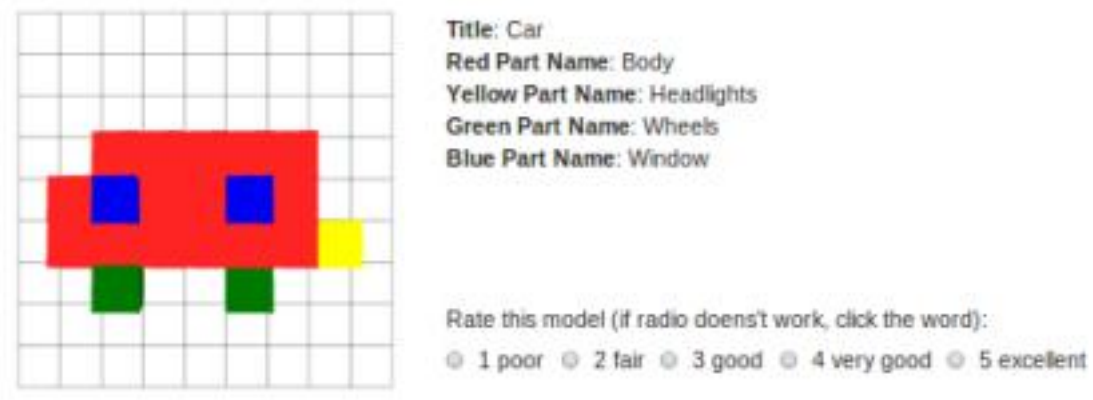

Figura 1.7: Interface web para coleta de dados.

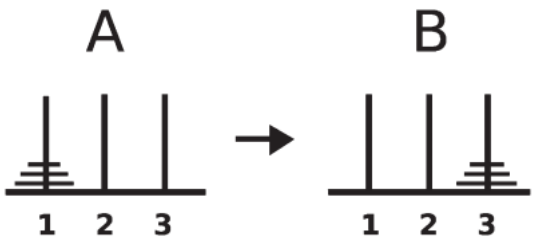

Figura 1.8: Problema da *Hanoi Towers*.

ROS se baseia em um servidor central, *rosmaster*, no qual são conectados vários processos, que podem estar em computadores diferentes, funcionando numa topologia ponto-à-ponto. Dessa forma, é possível executar tarefas em tempos real; enquanto alguns publicam mensagens, outros leem e executam as tarefas designadas.

São suportadas quatro linguagens distintas: C++, Python, Octave e LISP. Tal flexibilidade permite que o usuário escolha a que atender melhor suas preferências individuais, no robô foi utilizado quase que exclusivamente C++. Os tipos de mensagens são geradas a partir de pequenos arquivos de texto, a fim de facilitar a integração.

Possui diversos mecanismos para visualização, geração de gráficos, auto geração de documentos, entre outras ferramentas que podem ser usadas durante o desenvolvimento. Além de ser gratuito e de código aberto.

1.4 Rosbridge

Rosbridge possibilita a utilização de websockets para aplicações usando o ROS, permitindo que mesmo clientes que não o utilizam subscrevam, publiquem e chamem serviços utilizando mensagens ALEXANDER *et al.* (2012).

A grande vantagem da utilização do rosbridge é a independência de qual linguagem utilizar. Por não estar limitado ao ambiente do ROS pode ser utilizado qualquer linguagem que suporte WebSockets, podendo assim ser desenvolvido para qualquer plataforma como Windows, Android, IOS entre outras.

1.5 Estrutura do Texto

Este trabalho foi dividido em seis capítulos.

No capítulo 2 é descrito o robô com esteiras DIANE. Diz-se quais foram os sensores utilizados, o computador embarcado e como é feita a comunicação.

No capítulo 3 é explicado como foi organizado o software, quais processos que rodam no computador embarcado e quais devem rodar no computador remoto, além do que é necessário para operar o DIANE e qual é a utilidade de cada pacote.

É explicado e mostrado, no capítulo 4, o que é preciso para a criação de uma interface web para robôs usando a biblioteca *roslibjs*.

O capítulo 5 apresenta a interface desenvolvida para o robô DIANE e todas as suas funcionalidades, expondo para quê servem e como utilizar cada parte da interface.

A explicação sobre o código, como foi implementado, sua divisão, principais funções e variáveis está descrito no capítulo 6.

No último capítulo, de número 7, é apresentada a conclusão tirada acerca do trabalho, além de outros previstos para serem desenvolvidos no futuro.

Capítulo 2

Descrição do robô com esteiras DIANE

O DIANE é um robô com esteiras, desenvolvido no Laboratório de Controle da COPPE/UFRJ. O robô foi desenvolvido para manipular materiais perigosos de forma segura. Seu controle é realizado por um operador a uma distância segura, esse tipo de robô é conhecidos como *Explosive Ordnance Disposal*(EOD).

O robô possui duas esteiras laterais e dois pares de braços. Para o deslocamento são usadas as esteiras e, a depender da posição dos braços, esses também são utilizados. Além de auxiliar no movimento, são usadas para transposição de obstáculos.

Como dito anteriormente, os movimentos linear e angular são gerados pela associação das esteiras laterais. Caso a velocidade de ambas seja a mesma e tenham mesmo módulo e sentido, o movimento gerado será somente linear. Caso possuam mesmo módulo, mas sentidos opostos, o robô irá girar em torno de seu próprio eixo. Para outros casos, o robô terá movimento linear($v(t)$) e angular(ω) associados de acordo com a equação (2.1) DE LIMA (2016), no qual R é a distância entre o centro do robô e o da esteira, $\dot{\phi}_d$ é a velocidade linear da esteira direita e $\dot{\phi}_e$ é a velocidade linear da esteira esquerda.

$$\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2R} & -\frac{1}{2R} \end{bmatrix} \begin{bmatrix} \dot{\phi}_d \\ \dot{\phi}_e \end{bmatrix} \tag{2.1}$$

Os dois pares de braços do robô têm como principal objetivo a transposição de obstáculos, sendo um par dianteiro e outro traseiro. Cada par está conectado pelo mesmo eixo, desta forma um par executa o mesmo movimento, ou seja sempre terão a mesma velocidade e posição do seu par.

Tabela 2.1: Especificações dos motores

Especificação/Motor	Maxon EC 45	Maxon EC 32
Tensão Nominal	48V	12V
Velocidade Sem Carga	6160 rpm	15100 rpm
Corrente Sem Carga	244 mA	662mA
Velocidade Nominal	5490 rpm	13400 rpm
Torque Nominal	347 mNm	44.4 mNm
Corrente Nominal	4.86 A	6.51 A
Torque protocolar	3530 mNm	428 mNm
Corrente Protocolar	47.7A	57.2 A
Máxima eficiência	86 %	80 %
Resistência Terminal	1.01Ω	0.21Ω
Indutância Térmica	$0.448mH$	$0.03mH$
Constante de Torque	$73.9mNm/A$	$7.48mNm/A$
Constante de Velocidade	$129rpm/V$	$1280rpm/V$
Gradiente de Velocidade/Torque	$1.76rpm/mNn$	$35.8rpm/mNm$
Constante de tempo mecânico	$3.85ms$	$7.48ms$
Inercia do Rotor	$209gcm^2$	$20gcm^2$
Tipo do rolamento	$ballbearings$	$ballbearings$
Máxima Velocidade	$120000rpm$	$25000rpm$
Jogo Axial	$0 - 0.15mm$	$0 - 0.14mm$
Máximo carregamento Axial (Dinâmico)	$20N$	$5.6N$
Eixo suportado (estático)	$170N$	$98N$
Máximo carregamento radial	$180N$	$28N$
Numero de polos pares	1	1
Numero de fases	3	3
Numero de ciclos de autoclave	0	0
Proteção	$IP54$	$-$
Redução GearBox	$GP52C$	$GP32C111:1$
Peso	$1100g$	$270g$
Encoder	$-$	$HEDL-5600A06$
Resolução do encoder	$-$	$500cpr$

2.1 Hadware dos motores

O DIANE possui quatro motores, sendo um para cada esteira e outro para cada par de braços. Os das esteiras são do modelo *Maxon EC45*, enquanto os dos braços são do *Maxon EC32*, suas especificações são mostradas na tabela 2.1. Ambos possuem sensores de efeito Hall e enconders para medir a posição angular dos motores e realizar o controle dos mesmos.

São usados quatro *drivers* de potência para controlar os movimentos de cada um dos motores. Os drivers utilizado são as EPOS (Maxom Motor) que possibilitam o controle dos motores por três modos: velocidade, posição e corrente, no entanto, o DIANE usa apenas posição e velocidade.

A unidade de velocidade usada pelo *driver* é de rotações por minuto (RPM).

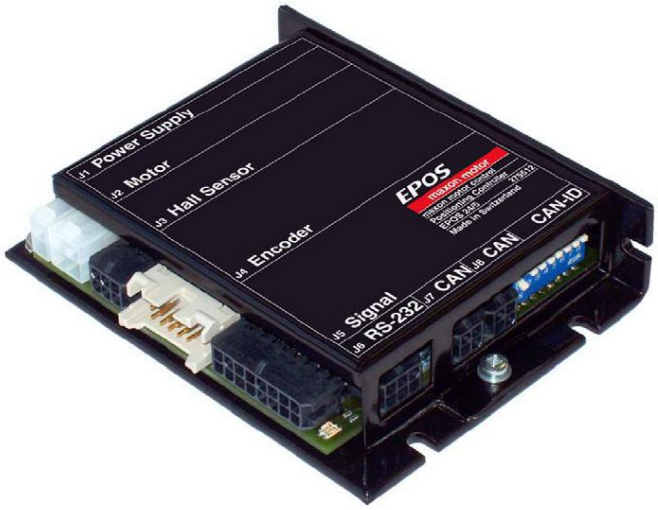

Figura 2.1: Epos modelo 24/5. Fonte:https://www.maxonmotor.com/

Existem algumas configurações que devem ser feitas para o bom funcionamento das EPOS no controle por velocidade, como os parâmetros do controlador proporcional integral(PI), limites de corrente, velocidade e aceleração máxima. Essas configurações devem ser feitas usando uma ferramenta disponível pela fabricante. A comunicação com as EPOS é feita através do barramento da rede CAN(*Controller Area Network*).

Por último, o controle por posição utiliza um *encoder*. Nesse, seus parâmetros são configurados na EPOS e usados para determinar a posição do motor a partir da soma dos pulsos do *encoder*, das características do eixo do motor e de sua rotação. O *driver* também possui um controlador proporcional integral derivativo (PID) no qual os ganhos devem ser ajustados conforme o desejado.

2.2 Sensores utilizados para mapeamento do ambiente

O DIANE possui dois sensores para mapear o ambiente, um deles é o *kinect*, o qual é utilizado como câmera durante a teleoperação, para medir o *pitch* e detectar escadas. O outro, *laser scan*, é usado para o mapeamento do ambiente, dando noção de distância para objetos ao seu redor.

O *kinect* é um dispositivo RGB-D (*Red Green Blue - Depth*) desenvolvido pela *Microsoft* juntamente com a *Prime Sense*. Ele possui um projetor de infravermelho, uma câmera RGB e outra infravermelho, além de um acelerômetro. A comunicação é feita através de uma porta USB (Universal Serial Bus). Suas características são mostradas na tabela 2.2, o sensor *kinect* é mostrado na figura 2.2.

Tabela 2.2: Informações do Kinect Fonte: https://msdn.microsoft.com/en-us/library/

Alimentação	12 V (DC)
Campo de visão (Vertical)	43°
Campo de visão (Horizontal)	57°
Alcance de inclinação	±27°
Taxa de quadros	30 FPS
Características do acelerômetro	**3-axial configurado para alcance de 2G**
Acurácia do acelerômetro	1°

Figura 2.2: Sensor *kinect* usado para obtenção dos dados.

Para se obter dados da distância de objetos próximos foi utilizado o laser UST-10LX, da Hokuyo. Um sensor pequeno e leve que utiliza comunicação TCP/IP através de um cabo Ethernet conectado à rede ethernet/wi-fi do DIANE. Suas características são apresentadas na tabela 2.3.

Figura 2.3: Hokuyo UST-10LX Fonte: http://hokuyo-aut.jp

2.3 Computador embarcado

Para gerenciar os componentes embarcados, foi usada a placa ADLQM67PC, desenvolvida pela *ADL Embedded Solutions* e mostrada na figura 2.4. Ela segue o padrão PC104 que permite o empilhamento de outros de dispositivos na horizontal,

Tabela 2.3: Informações do laser UST-10LX Informações retiradas do manual

Alimentação	12/24 V (DC)
Ângulo de varredura (Horizontal)	270°
Resolução ângular	0.25°
Alcance	0.06m - 30m
Período de varredura	25 ms

reduzindo o espaço necessário, se comparado ao de computadores convencionais. Foi usado um processador Intel i7 da segunda geração, com 4GB de memória RAM e um SSD de 256GB.

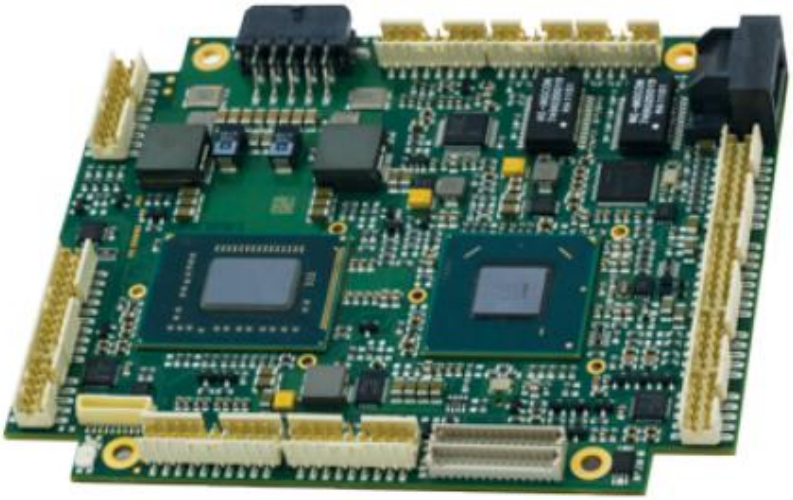

Figura 2.4: Placa ADQM67PC.

O computador está conectado com a placa CAN PCI104/200, que faz a interface com a rede CAN, que é usada para comunicação com os *drivers*, como dito anteriormente. Além disso, possui interfaces *Ethernet* e *USB*, que são usadas para comunicação com outros dispositivos rede interna e sensores, respectivamente.

O sistema operacional escolhido foi o Ubuntu, uma distribuição Linux. Tal escolha foi por ser um sistema robusto, com grande comunidade colaborativa, acessibilidade e compatibilidade com o *framework ROS*, usado para o desenvolvimento do software do DIANE.

Para a elaboração e organização dos códigos foi escolhido o *QtCreator* como ambiente de desenvolvimento. Apesar do ROS possuir ferramentas gráficas, estas não foram utilizadas.

Os comandos para as EPOS são passados por uma rede CAN pelo computador embarcado. Há um protocolo para essa rede chamado *CANOpen*, a fim de garantir a entrega de pacotes com rigoroso tratamento de erros e recebimento de mensagens.

As mensagens enviadas possuem cabeçalhos para identificação de qual EPOS se remete ou destina e qual o tipo de mensagem que está sendo enviada.

2.4 Comunicação

São usados dois protocolos de comunicação embarcados no DIANE: TCP/IP e CAN. O TCP/IP é usado na comunicação entre os nós do ROS e entre os computadores. É utilizado um roteador, modelo 195e da *ESTeem* e que pode ser visto na figura 2.5, com Wi-Fi ligado ao *switch* conectado no computador embarcado.

Figura 2.5: Roteador Wi-Fi utilizado.

São transmitidos pela rede CAN periodicamente mensagens com valores de *set-point*, comandos e dados, esse tipo de dado é conhecido como PDO(*Process Data Object*). Esse tipo de comunicação segue o principio de produtor-consumidor, no qual a mensagem é transmitida por um nó produtor e recebida pelos nós consumidores. Uma maneira eficiente de transmissão de mensagem, principalmente quando são transmitidas varias mensagens ao mesmo tempo.

Capítulo 3

Organização do Software

Nesse capítulo será apresentada a distribuição do software e qual a responsabilidade de cada nó, além de tudo necessário para a opera-los no robô DIANE.

Para o procedimento remoto, demanda-se um computador que funcione como interface para o operador, o qual se comunica com o computador embarcado através de uma conexão sem fio *Wi-Fi*, conforme explicado no capítulo 2.

O computador embarcado executa as rotinas de *ROS* necessárias para a comunicação e controle de seus componentes, como sensores e atuadores, fato elucidado detalhadamente em AZEVEDO (2017).

3.1 Nós Embarcados

Os nos do DIANE podem ser separados em:

- Sensoriamento
- Atuação
- Comunicação

Os processos de sensoriamento são os relacionados aos sensores, como o laser e o *kinect*. Os processos de atuação são os que cuidam dos motores, os quais são controlados pelas quatro EPOS.

Nesta secção serão mostrados os principais pacotes que rodam no computador embarcado. Em geral, são aqueles que se comunicam diretamente com os componentes do DIANE, como as EPOS e detecção de escadas.

3.1.1 Pacote EPOS

Este pacote é responsável pelas informações relacionadas aos drives de controle dos motores. Ele realiza a troca de informações entre computador e *driver*, recebendo

a velocidade e posição de cada EPOS, além disso, o pacote deve calcular e enviar os valores de correntes desejados para que a EPOS esteja na posição ou velocidade desejada.

3.1.2 Pacote Rosbridge

O pacote *rosbridge* possibilita a comunicação do ROS com *HTML5 websockets* ou TCP/IP *sockets*. Dessa forma, é possível publicar, subscrever e requisitar serviços através de objetos *JSON(JavaScript Object Notation)* LEE (2012), um modelo usado para armazenamento e transmissão de informações no formato texto muito utilizado em aplicações web.

Utilizando o pacote *roslibjs* (uma biblioteca desenvolvido em JavaScript) para interação com o ROS através do navegador FOUNDATION (2018). Dessa forma é possível esconder a complexidade do *ROS* e desenvolver uma aplicação web. Isso torna a criação das aplicações mais simples e expande as possibilidades para a utilização dos robôs.

3.1.3 Pacote DIANE Octomap

O pacote DIANE Octomap é o responsável por encontrar a escada, modelá-la e passar os parâmetros, angulação e altura da escada, para o controle de subida. A detecção é chamada através de um serviço e necessita que o *kinect* ou o laser estejam ligados para obtenção da nuvem de pontos. Caso seja encontrada alguma escada, o serviço retorna uma mensagem com os dados da mesma. Todo esse processo é esmiuçado em CHAN (2017). Na figura 3.1 podemos ver uma escada detectada, em vermelho, utilizando esse pacote.

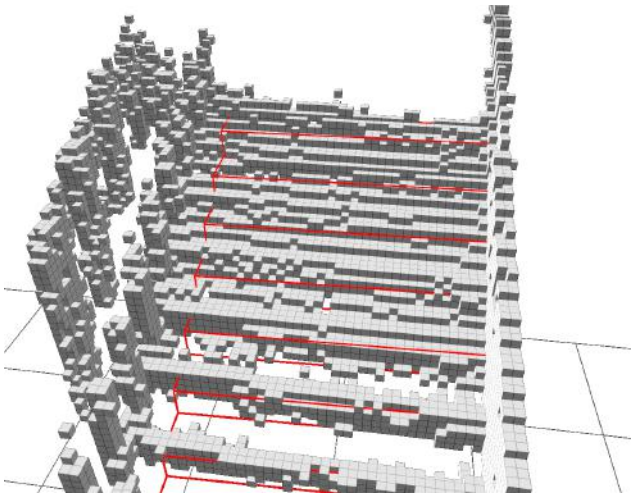

Figura 3.1: Exemplo de escada detectada pelo pacote DIANE Octomap.

3.1.4 DIANE Controller

O pacote DIANE *Controller* é o responsável por receber os valores de velocidade e posição desejados do robô, calcular o valor no qual cada esteira deve estar para atingir o valor desejado e então transmiti-lo para o pacote EPOS, juntamente com o modo de controle desejado.

3.1.5 Controle para subida de escada

Com a intenção de facilitar a teleoperação, foi desenvolvida uma metodologia para que a subida de escada funcionasse de forma autônoma, isto é, o robô completa a ação sem precisar de auxílio do operador. No entanto, caso desejasse, o operador poderia assumir o controle da velocidade linear e os braços seriam ajustados automaticamente.

São feitas duas transições durante o processo: a passagem do piso inferior para a escada e a saída da mesma para o piso superior. Além disso, há também o deslocamento entre os degraus. O método desenvolvido é baseado em Lima DE LIMA (2016).

O controle de subida foi feito para funcionar como uma máquina de estados. Sendo assim, cada estado possui uma tarefa e, de acordo com o estímulo que ele recebe, ocorre a mudança para outro estado.

Para a iniciação e utilização do controle é necessário que o robô esteja em frente à escada e com os valores modelados.

Ao iniciar o controle, o braço dianteiro é levado para a posição de 50° e o traseiro para 0°. Desse modo, o robô começa a se mover em direção à escada até subir os primeiros degraus e estar na angulação da mesma.

Com o robô posicionado, ele arruma seu braço dianteiro para a posição 0°, a fim de aumentar a área de contato com a escada, e então volta a se movimentar e subi-la. Caso não tenha nenhuma interferência, ele irá até o final dela e tombará. O controle encerra a subida ao atingir o nível do solo.

Para impedir o baque, é necessário que o operador inicie a saída suave da escada. Dessa forma, ele levará o braço dianteiro para a posição de $-90°$, o robô não tombará ao sair e, ao final, irá voltar o braço para a posição de 0°.

Os estados e as transições podem ser vistas na figura 3.2.

3.1.6 Inicialização do robô

Para iniciar os processos, são configurados arquivos no formato launch, esses arquivos costumam trazer um conjunto de nós que precisam rodar de forma conjunta para se obter a funcionalidade desejada. Para iniciar o robô é necessário executar alguns

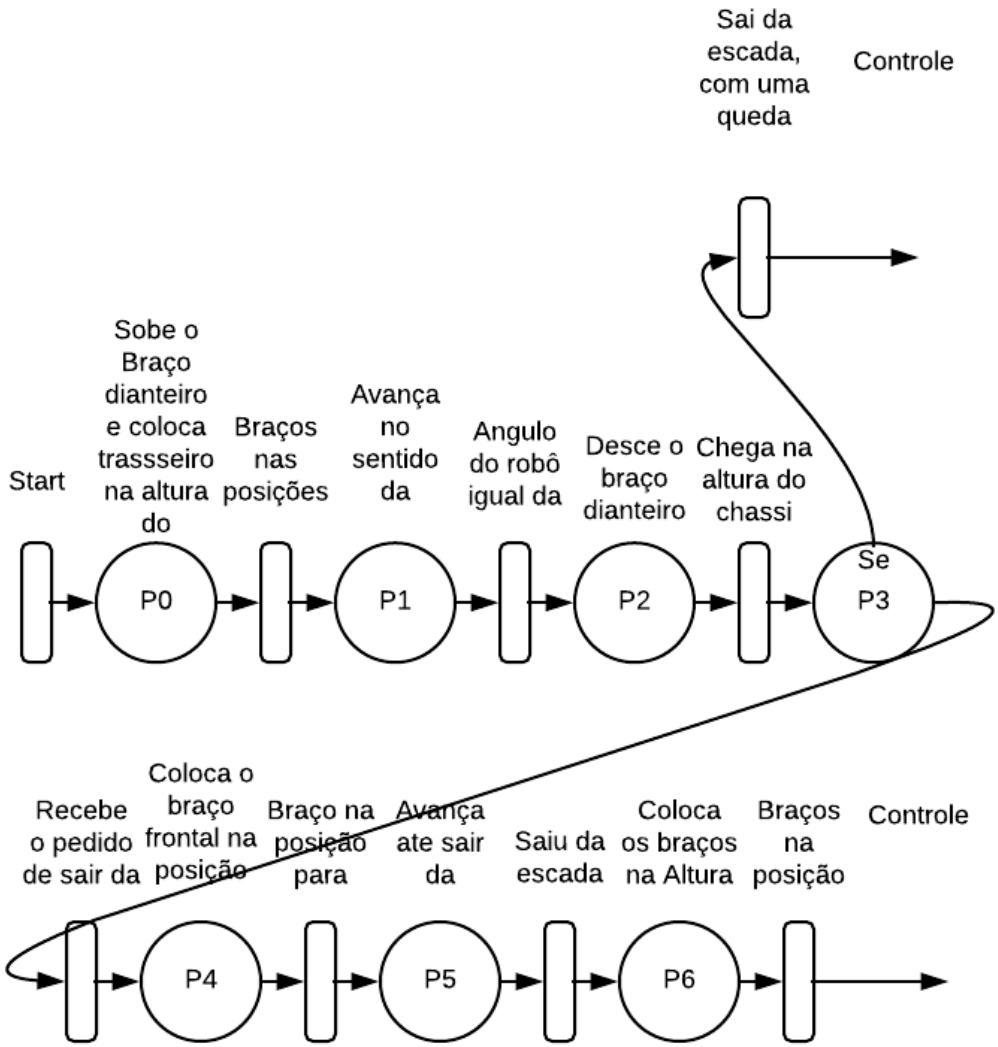

Figura 3.2: Máquina de estado que representa o controle de subida de escada.

desses arquivos, através dos seguintes comandos:

- roslaunch rosbridge_server rosbridge_websocket.launch
- roslaunch diane_files robot.launch
- roslaunch diane_files laser.launch

Para executar esses comandos deve se estar na mesma rede do robo e conecta se ao computador através do comando: ssh -v diane@172.16.11.27. Apos a execução desses o robô esta pronto para ser operado pela interface web. Os arquivos launch se encontram no apêndice A.

Capítulo 4

Interface Web para Robôs

Apesar de contar com inúmeras vantagens e facilidades, o sistema ROS possui alguns problemas relacionados à usabilidade e acessibilidade. Oficialmente, o ROS possui suporte somente para o sistema operacional Ubuntu. Ainda que alguns projetos ofereçam possibilidade de instalação em outros sistemas, como OS X, eles apresentam algumas instabilidades TORIS *et al.* (2015).

A criação de uma interface Web para monitoramento e utilização do robô tem como principal objetivo poder acessá-lo de outros sistemas, não sendo necessário a instalação do *framework* ROS, permitindo uma maior usabilidade para o usuário.

Para a criação de uma página web que utiliza o ROS como *back-end* é necessária a comunicação entre ambos. Sendo assim, será apresentado neste capítulo como é feita essa comunicação e como usá-la para ler ou escrever em tópicos e chamar serviços baseados no projeto *Robot Web Tools* TORIS *et al.* (2015).

A comunicação entre o sistema ROS e o computador é dada através do protocolo estabelecido pelo *rosbrige*, no qual comunica-se usando mensagens do tipo JSON e *WebSocktes*. Com isso a comunicação pode ser feita utilizando navegadores de internet, não sendo necessária nenhuma instalação.

A principal biblioteca usada neste trabalho foi a *roslibjs*, que é própria para comunicação com os tópicos, serviços, entre outras funcionalidades do ROS. Neste capítulo, serão apresentados e exemplificados casos aplicados para a elaboração da interface e como utilizar esta biblioteca. Na figura 4.1 vemos como ficou a página desenvolvida no projeto.

4.1 Conexão com o ROS

O primeiro passo a ser dado é conectar-se ao servidor do *rosbridge*. Para isso, é necessário saber o IP ao qual deseja conectar-se e utilizar o código, mostrado na Listagem 4.1, em que o servidor está rodando no computador embarcado do DIANE, na porta 9090. Com esse código, é criada a conexão com o servidor *rosbridge*.

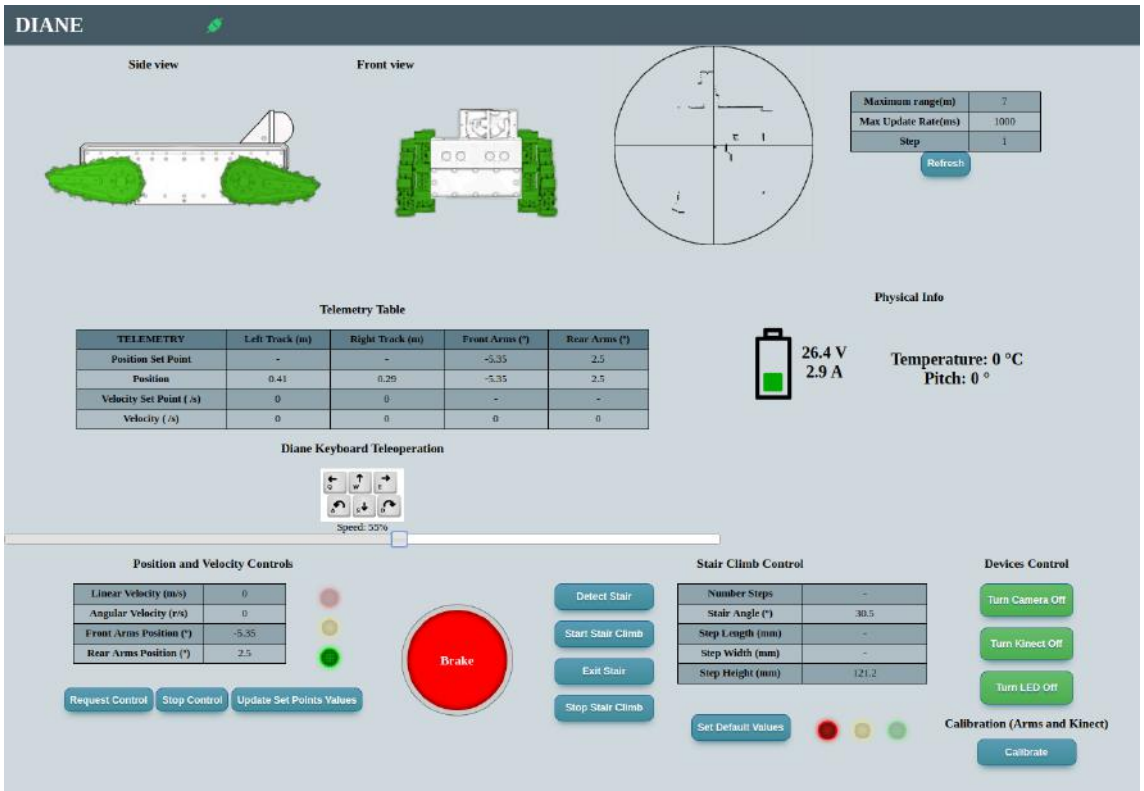

Figura 4.1: Página web desenvolvida para o acompanhamento do robô.

Listing 4.1: Exemplo de conexão.

```
var ros = new ROSLIB.Ros({
    url: 'ws://172.16.11.27:9090/'
});
```

Caso ocorra algum erro durante a operação, é possível que a conexão seja impedida ou interrompida. Por isso, durante o processo, é necessário que a conexão seja verificada para que o operador saiba se está conectado ao robô. Para tal fim, são usadas as funções mostradas em 4.2, as quais avisam se a conexão foi fechada ou se houve alguma falha.

Listing 4.2: Verificando se houve erro na conexão ou se ela foi encerrada.

```
ros.on('error', function(error) {
    console.log('Error connecting to websocket server: ', error);
    connectState = false;
});

ros.on('close', function() {
    console.log('Connection to websocket server closed.');
    connectState = false;
});
```

4.2 Trabalhando com tópicos

Os tópicos de ROS servem para receber e transmitir mensagens. Aqui seria bom lembrar da modularidade. Para quem lê, não importa quem envia. E para quem envia, não importa quem lê. os tópicos podem ser lidos ou escritos simultaneamente por agentes diferentes (interface web ou o robô).

4.2.1 Leitura de tópicos

Umas das principais funcionalidades da interface é mostrar as informações do robô de todas as naturezas, tais como: velocidade das esteiras, posição dos braços, temperatura do computador, quantidade de carga nas baterias, entre outras. Para obter essas informações, é necessária a leitura dos tópicos que as contêm.

Para a leitura é preciso, primeiramente, estar conectado ao *rosbridge*. Então, deve-se saber o nome do tópico que deseja ler e o tipo de mensagem desta mensagem.

Tendo essas informações, deve-se criar o objeto com os parâmetros desejados. A exemplo do 4.3, no qual passamos o objeto que contém a conexão, chamado de ros, temos: o nome do tópico que desejamos ler, no caso, "/diane/epos1/info"e o tipo da mensagem, que conveio a ser "epos/Info".

Listing 4.3: Função usada para leitura do tópico info da epos.

```
var listener = new ROSLIB.Topic({
    ros: ros,
    name: '/diane/epos1/info',
    messageType: 'epos/Info'
});
```

Com o objeto criado, pode-se chamar a leitura do tópico conforme desejar, como no exemplo 4.4. Com isso, será lida a última mensagem publicada. Ao receber o resultado, a mensagem vem no formato JSON, que é transformado num objeto de JavaScript.

Listing 4.4: Função usada para leitura do tópico info da epos.

```
listener.subscribe(function(message) {
    if (message.velocity == null) {
        leftTrackVelocity = '-'
    } else {
        leftTrackVelocity = Math.round(message.velocity * 10) / 10

    }
    if (message.position == null) {
        leftTrackPosition = '-'
    } else {
        leftTrackPosition = Math.round(message.position * 10) / 10
```

```
        }

        listener.unsubscribe();
    });
```

Uma observação importante a fazer em relação à leitura de tópicos é sobre o tipo de componente que se deseja ler. Caso ele não seja base 64, será necessário fazer uma decodificação, como em componentes do tipo *int8*.

4.2.2 Escrita em tópicos

A ideia não é apenas conseguir observar os parâmetros do robô, mas também controlá-lo. Para isso, é necessário a escrita em tópicos.

Podemos ver no exemplo 4.5 que a criação do objeto para envio funciona da mesma forma que para a leitura, necessitando dos mesmos parâmetros.

Listing 4.5: Criação do objeto para envio de mensagem.

```
var controlMSG = new ROSLIB.Topic({
    ros: ros,
    name: '/diane/diane_controller/input',
    messageType: 'controller/Control'
});
```

Para o envio da mensagem, primeiramente, é necessária a criação da mesma. Nela, cada um dos componentes deve possuir nome e valor a serem separados de outros componentes por vírgula, vide exemplo 4.6. Com a mensagem gerada, podemos publicá-la através do objeto gerado anteriormente.

Listing 4.6: Criação da mensagem e envio.

```
var msg = new ROSLIB.Message({
    originId: control_id,
    modes: [1, 1, 1],
    data: [0, 0, 0, 0]
});
controlMSG.publish(msg);
```

4.3 Serviços no ROS

A leitura e escrita em tópicos é uma maneira de comunicação flexível, usualmente necessária em sistemas distribuídos. O serviço funciona como um par de mensagens: a requisição e a resposta. Com isso, o cliente pode enviar a solicitação da mensagem conhecendo o nome do serviço desejado e esperar pela sua resposta.

4.3.1 Trabalhando com serviços

Em relação à forma como se usa o serviço, não há grandes diferenças se comparada à maneira como é feita na leitura ou escrita de tópicos. A primeira exigência é a criação do objeto com as informações dos serviços requeridos, como no exemplo 4.7, no qual se deseja fazer uma requisição para obter um ID e, assim, controlar o DIANE. São tidos como parâmetros o objeto tendo efetiva conexão, o nome do serviço desejado e o tipo de serviço.

Listing 4.7: Criação da mensagem e envio.

```
var requestId = new ROSLIB.Service({
    ros: ros,
    name: '/diane/diane_controller/request_id',
    serviceType: 'controller/RequestID'
});
```

A requisição é feita como mostrada em 4.8, em que se chama o serviço e recebe o resultado. Ao obtê-lo, ele é tratado como um objeto JavaScript, na qual deve-se saber o elemento contido para acessá-lo. Neste caso, recebemos o ID e salvamos para usá-lo com o propósito de controlar o robô.

Listing 4.8: Criação da mensagem e envio.

```
requestId.callService(request_id, function(result) {
    control_id = result.id;
    console.log('id: ' + result.id);
});
```

No entanto, existem muitas outras bibliotecas e inúmeros projetos que podem ser feitos utilizando aplicações web para robôs. Existem, inclusive, bibliotecas para exibições 2D (ROS2DJS) e 3D (ROS3DSJ) que podem ser usadas, por exemplo, para mostrar as posições do robô no momento.

É importante salientar que existe um problema no código para *streamer* que impede a transmissão do vídeo ser feita de uma porta diferente da 9999 FOUNDATION (2018).

Capítulo 5

Interface Web para o Robô DIANE

A interface desenvolvida, mostrada na figura 4.1, traz várias informações aproveitadas durante a teleoperação do robô como funções para auxiliar na subida de escada e durante o percurso. Ela foi elaborada utilizando a linguagem de programação *JavaScript*. A principal biblioteca usada foi o *roslibjs*, que utiliza *WebSockets* para se conectar ao *rosbridge*, permitindo publicar, sobrescrever, chamar serviços, entre outras funcionalidades essenciais do ROS.

Para a criação da página, foi necessário que todo o ambiente do ROS estivesse configurado e funcionando. A partir disso, foi possível se comunicar através de websockets e mensagens JSON, e então desenvolver a interface.

Será explicado nesse capítulo para o quê servem, como ou quando utilizar cada uma das partes da interface.

5.1 Cabeçalho

No cabeçalho da interface temos o nome do robô, DIANE, e uma imagem de um conector, que mostra se a interface está ou não conectada a ele.

Quando conectada, o conector fica como mostrado na figura 5.2 e quando desconectada, como na figura 5.1. Ao iniciar a interface, ela fará uma tentativa de se conectar ao robô. Caso não consiga, o usuário deverá clicar na imagem para que a página tente novamente. Na hipótese de haver algum problema com a conexão, será mostrada no console do navegador a mensagem de erro.

Figura 5.1: Símbolo quando a página não está conectada ao DIANE.

Figura 5.2: Símbolo quando a página está conectada ao DIANE.

5.2 Corpo

5.2.1 Visão lateral

No corpo da interface, na primeira linha do lado esquerdo há a visão lateral, que apresenta uma representação do robô visto lateralmente. Essa imagem tem a intenção de mostrar a posição dos braços dianteiros e traseiros, além do modo de controle em que esses se encontram.

O modo de controle no qual o braço está é representado pela sua mudança de cor. O significado de cada cor é mostrado na tabela 5.2.1. Em relação à posição, a imagem acompanha o movimento real. Dessa forma, a angulação da imagem é igual a dos braços. Nas figuras 5.3 e 5.4 temos exemplos de variações do modo de controle e da posição.

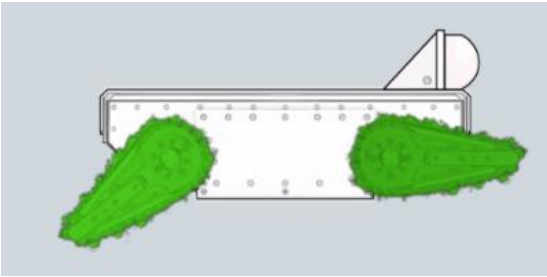

Figura 5.3: Braços do robô sendo controlados.

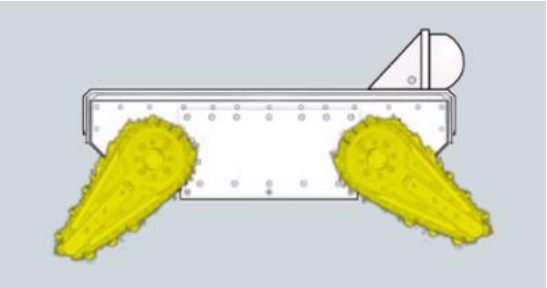

Figura 5.4: Braços em modo *Break*.

Tabela 5.1: Significado de cada cor dos braços e esteiras, nas partes da *visão frontal e lateral*

Cores da Visão Frontal & Lateral	
Cor	Significado
Branco	Não controlado
Vermelho	Estado de emergência
Amarelo	Estado de break
Verde	Sendo controlado

5.2.2 Visão Frontal

A visão frontal tem um objetivo parecido com a visão lateral. A diferença é que, além de apresentar por cores em qual tipo de controle os braços se encontram, essa representação também se estende para as esteiras, seguindo o mesmo padrão de cores da tabela 5.2.1. Nas figuras 5.5, 5.6 e 5.7 temos exemplos da visão frontal com o robô estando em diferentes modos de controle.

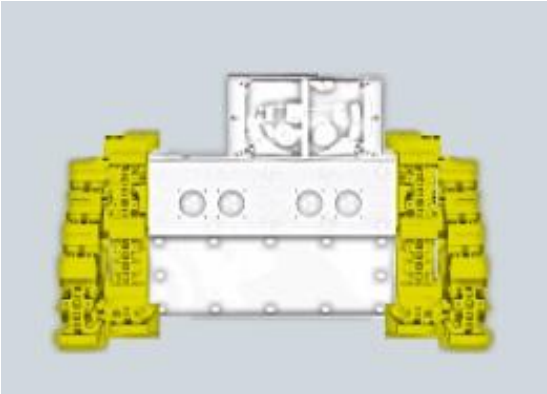

Figura 5.5: Esteiras e braços em modo *break*.

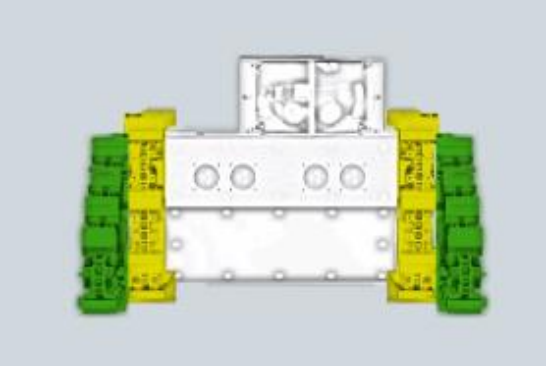

Figura 5.6: Braços sendo controlados e esteiras em modo *break*.

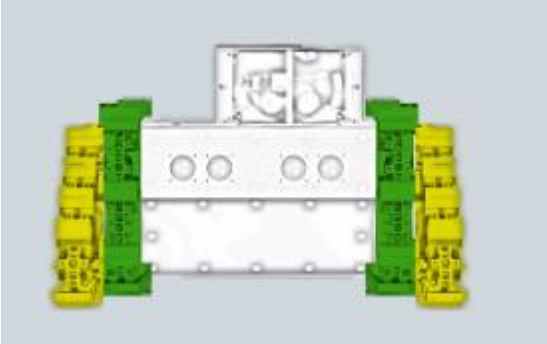

Figura 5.7: Braços em modo *Break* e esteiras sendo controladas.

Com a visão lateral e o visão frontal o operador sabe em qual modo de controle os motores estão e a posição dos braços. Dessa maneira, ele possui uma boa representação da forma em que o robô se encontra e o que está controlando no momento, evitando que confunda os comandos e cometa algum erro que possa prejudicar a navegação.

5.2.3 Laser Scan

O laser Scan é localizado ao lado da visão frontal, mostrado na figura 5.8, usando os dados obtidos pelo laser hokuyo tem-se a visão de objetos ao redor do robô. Junto ao laser há uma tabela para ajuste de seus parâmetros que são:

- Maximum Range: É o maior valor que será mostrado, equivalente as bordas do laser Scan, é utilizado para controlar o Zoom.
- Max Update Rate: O valor da maior taxa de atualização do laser, caso a taxa seja muito baixo pode ocorrer lentidão na interface devido a grande quantidade de atualizações.
- Step: É utilizado para diminuir a quantidade de pontos que são exibidos, caso a resolução do laser seja alta é possível diminuir a quantidade de pontos mostrado diminuindo a chance de que ocorra lentidão na interface.

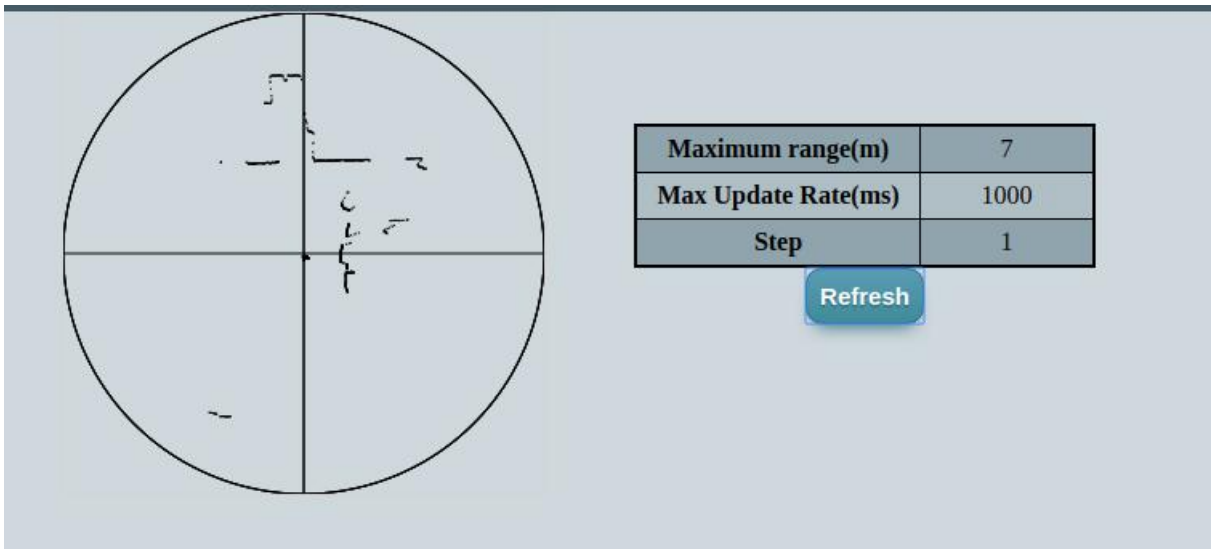

Figura 5.8: Radar desenvolvido para o site.

5.2.4 Tabela de telemetria

A tabela de telemetria é encontrada logo abaixo da visão lateral. Ela mostra os valores de velocidade, a posição e o *setpoint*. As posições e velocidades são sendo eles expressos em metros e metros por segundo para as esteiras e em graus e graus por segundo para os braços, respectivamente. O mesmo vale para o *setpoint*. O controle de uma parte do robô não pode estar em dois modos distintos ao mesmo tempo, sendo assim, será mostrado o valor do modo atual e o outro terá um hífen(-), indicando que não está sendo utilizado.

Na figura 5.9 vemos um exemplo da tabela sendo usada no modo de controle não controlado. Dessa forma, não há valor de *setpoint* para nenhum dos casos.

Telemetry Table

TELEMETRY	Left Track (m)	Right Track (m)	Front Arms (°)	Rear Arms (°)
Position Set Point	-	-	-	-
Position	0.17	0.06	0	0
Velocity Set Point (/s)	-	-	-	-
Velocity (/s)	0	0	0	0

Figura 5.9: Tabela de telemetria com dados de posição e velocidade.

5.2.5 Informações físicas

Ao lado da tabela de telemetria temos as informações físicas do robô que são a bateria, temperatura do computador interno e o *pitch* do robô. A temperatura mostrada é a do núcleo mais quente, tendo sido recebida a informação da temperatura de todos os núcleos, através da biblioteca *libsensor_monitor*. O valor do *pitch* é obtido pelo acelerômetro do *kinect*, enquanto a das bateria vem a partir da tensão e corrente que é lido pela EPOS. Podemos ver as informações físicas na figura 5.10.

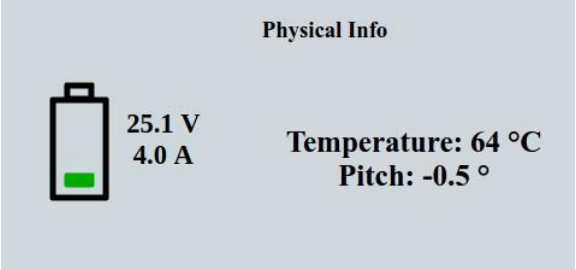

Figura 5.10: Informações físicas do robô.

5.2.6 Tabela de controle

No canto inferior esquerdo encontra-se a tabela de controle, ferramenta a qual é usada quando se deseja controlar o robô através da interface. As esteiras são controladas por velocidade e os braços pela posição.

Para utilizá-la é preciso, primeiramente, requisitar o controle. Isso é feito clicando no botão *Request Control*. A partir disso, o semáforo ao lado mudará para amarelo, indicando que a interface está requisitando o controle. Caso já exista alguém controlando o robô, ele irá esperar o acesso ser liberado para assumir. Ao conseguir o semáforo acenderá a luz verde. Na tabela 5.2.6 podemos ver o significado de cada estado.

É possível controlar o DIANE através da interface por duas maneiras:

A primeira é utilizando a tabela de controle, na qual deve-se escolher a velocidade linear e angular das esteiras em metros por segundo, (*Linear Velocity* e *Angular Velocity*), sendo que é possível passar valores negativos dependendo do sentido desejado. O controle dos braços é feito por posição, devendo-se escolher o valor desejado da mesma em graus.

A segunda maneira é utilizando o teclado. As teclas de comandos são mostradas na tabela 5.2. Os dois modos podem ser utilizados simultaneamente para isso basta ativar o controle pelo teclado utilizando o botão *Turn Keyboard On*. Dessa forma o controle da tabela continuará a ser enviado e quando o teclado ser usado o comando será adicionado ao controlo enviado.

Tabela 5.2: Tabela de controle pelo teclado

Tecla	Comando
W	Andar para frente
S	Andar para trás
A	Andar para direita
D	Andar para esquerda
E	Descer Braço direito
Q	Subir Braço direito
Z	Descer Braço esquerdo
C	Subir Braço esquerdo

Uma observação importante é que, enquanto não está sendo usado, a tabela atualiza seus valores para os atuais do robô. Dessa forma, quando assumido o controle, estarão padronizadas as velocidades linear e angular iguais à zero e as posições do braço estarão análogas às do robô.

Quando não se desejar mais o controle através da interface, basta selecionar *Stop Control*. Isso fará com que retorne para o estado não controlado e o sinal fique vermelho novamente. Como só é permitido que o controle seja feito por um dispositivo por vez é necessário desligar o controle para utilização de outro, como por exemplo pelo joystick.

Tabela 5.3: Significado do sinal presente na região do controle de velocidade e posição.

Sinal de controle da subida de escada	
Cor do semáforo	Significado
Vermelho	Controle não está ativo
Amarelo	Requisitando o controle
Verde	Controlando o robô

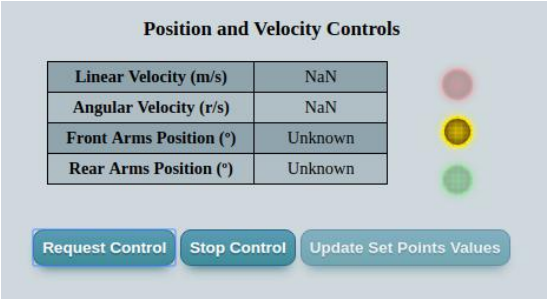

Figura 5.11: *Position and Velocity Controls* após a requisição do controle.

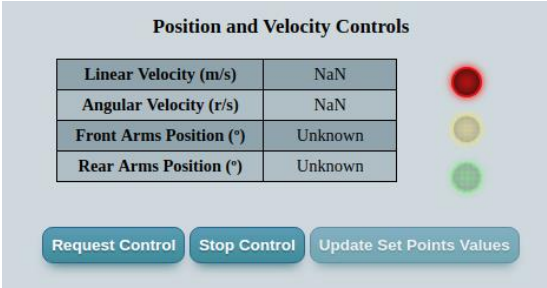

Figura 5.12: *Position and Velocity Controls* ao ser iniciada.

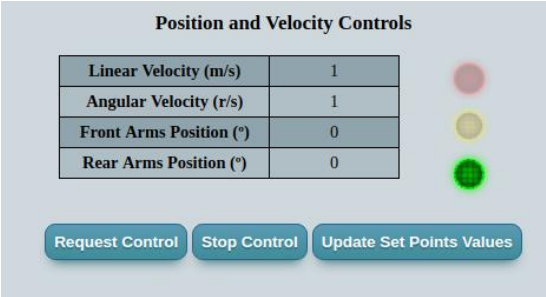

Figura 5.13: *Position and Velocity Controls* com controle ativo.

5.2.7 Break

Ao lado da tabela de controle, existe um botão *break*. Caso a interface esteja com o controle, o botão estará ativo, como mostrado na figura 5.14. Se selecionado, o robô entrará em modo *break*, enviando um controle de velocidade zero fazendo com que todos os motores fiquem travados, impedindo tanto o movimento das esteiras quanto o dos braços. Para sair do modo *break* é necessário clicar novamente no botão, que estará como na figura 5.15.

5.2.8 Controle para subida de escada

Stair Climb Control é referente ao controle autônomo da escada, o qual é mostrado na figura 5.16. A fim de iniciar o controle autônomo, é necessário que se tenha os parâmetros da escada. Para isso, deve-se posicionar o robô em frente à ela e clicar no botão *Detect Stair*. Esse chamará o serviço para detecção de escadas, *DIANE Octomap*, e receberá como resposta os parâmetros da mesma, caso alguma seja encontrada. Outra opção seria utilizar o botão *Set Default Values*, o qual usa valores comuns à maioria das escadas, ou escolher os parâmetros desejados colocando-nos na tabela.

Figura 5.14: Botão de *break* passível de ativação.

Figura 5.15: Botão ativo no modo *break*.

A seguir, o usuário poderá clicar no botão *Start Stair Climb* e iniciar o processo da subida. Ela será feita de forma autônoma, a menos que se queira controlar a velocidade linear, o que demandaria requisitar o controle da mesma pelo *joystick*.

Para uma saída suave, deve-se clicar no botão *Exit Stair*. Caso contrário, o robô irá andar até o final da escada e sofrerá um tombamento, conforme explicado no capítulo 4.

Caso queira interromper o controle de subida, é necessário clicar no botão *Stop Stair Climb*, encerrando o processo.

O semáforo serve para demostrar em qual estado está o controle, estando o significado de cada cor mostrado na tabela 5.2.8.

Figura 5.16: Tabela de telemetria com dados de posição e velocidade.

5.2.9 Controle de dispostivos

O controle de dispositivos possui botões que servem para ligar componentes conectados as portas lógicas da EPOS, como o *kinect*, LED entre outros, os quais não ficam ativos a todo instante, a fim de economizar bateria. Os componentes podem

Tabela 5.4: Significado do sinal presente na região do controle referente à subida da escada.

Sinal do controle de subida da escada	
Cor do semáforo	Significado
Vermelho	O controle está desligado
Amarelo	O controle está ligado, porém pausado
Verde	O controle está ligado

ser ligados ou desligados quando oportuno, usando o botão referente ao dispositivo desejado. Podemos ver esses botões na figura 5.17.

Figura 5.17: Botões para ligar e desligar dispositivos externos.

5.2.10 Calibração

A calibração foi criado para possibilitar a calibragem da posição dos braços. Quando o robô é ligado, é considerado que as posições estão em zero, no entanto elas, estarão na forma que foram deixados. Para corrigir isso, deve-se ligar o robô, colocar seus braços na posição zero e clicar no botão *calibrate*, mostrado na figura. Dessa forma, ele será redefinido para a posição zero correta.

O botão não pode ser utilizado enquanto o controle estiver ativo pela interface. Caso o usuário tente operá-lo nesse momento, aparecerá um alerta informando que o controle está ativo e, por isso, a calibração não funcionou.

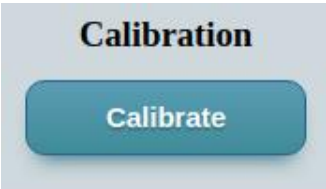

Figura 5.18: Botões para calibrar a posição dos braços.

Capítulo 6

Organização do código da Interface Web

Nesse capítulo será explicado o código desenvolvido. As linguagens utilizadas para o desenvolvimento da aplicação web foram o JavaScript, HTML e CSS, sendo os dois últimos aqueles que fazem a explicação dos elementos da página, o design, e o JavaScript responsável pela dinâmica da mesma, consumindo e publicando os dados do ROS e os exibindo na tela.

Para melhor organização, o código está dividido em vários arquivos, os quais serão explicados separadamente, bem como os significados das variáveis, funções e como é feita a comunicação entre todos.

6.1 CSS

O arquivo main.css é onde está estabelecido todo o estilo da página. Ele serve para definir o formato, o local da página que será exibido cada elemento e o espaço demarcado para cada um. Isto é, tudo aquilo que é relacionado ao design da página consta neste arquivo.

A ideia para organizar a página é criar divisões maiores que ocupem uma faixa completa da página, e subdivisões para cada elemento, todas essas divisões são chamadas de *div* e recebem um id relativo as configurações definidas no CSS. Podemos ver um exemplo dessas configurações em 6.1, essa utilizada pelo visão lateral do robô, define que ela deve se posta ao lado esquerdo da tela utilizando 25% da largura e todo a altura disponível, são utilizados valores relativos para que a Interface se ajuste à resolução e tamanho de cada tela.

Listing 6.1: Configuração utilizada para a visão lateral do robô.

```
#robot_side_view {
float: left;
```

```
width: 25%;
height: 100%;
}
```

6.2 HTML

O HTML é o arquivo que une todos os arquivos para exibição. Ao ser aberto, ele carrega todos os arquivos JavaScript utilizados para o desenvolvimento da interface e os exibe posicionando de acordo com as configurações feitas no arquivo de CSS.

Para a criação de cada setor são utilizadas as *div* e o id relativo a configuração css desejada, podemos ver um exemplo em 6.2 no qual é feito o posicionamento da visão lateral na interface. A imagem da visão lateral está dividida em três partes: os dois braços e o corpo do robô. Sendo assim temos a div principal, robot_side_view, que contem as três div, robot_side_main_image , robot_side_rear_arm_region e robot_side_front_arm_region, sendo que cada uma dessas possui a tag da imagem correspondente criando assim a imagem que é exibida na página, mostrada na figura 5.3.

Listing 6.2: Posicionamento da imagem relativa a visão lateral.

```
<div id=robot_side_view>

        <h3>Side view</h3>

        <div id='robot_side_main_image'>
                <div id='robot_side_body_region'>
                <img id='robot_side_main_body' src="img/body.png"
                alt="HTML5 Icon">
        </div>

        <div id='robot_side_rear_arm_region'>
                <img id='robot_side_rear_arm'
                src='img/rearArmNotControlled.png' alt='HTML5 Icon'>
        </div>

        <div id='robot_side_front_arm_region'>
                <img id='robot_side_front_arm'
                src='img/frontArmNotControlled.png'      alt='HTML5 Icon
                    '>
        </div>
</div>
```

Algumas funções são chamadas de forma recursiva como por exemplo as de atualizações, as que buscam informações do ROS, entre outras. Para isso, é usado função

setInterval que recebe como parâmetros a função que deve ser chamada juntamente com o período, um exemplo é mostrado em 6.3 no qual são chamadas as funções *updateRearArmImage* e *updateFrontArmImage* a cada 300 milissegundos. Esse procedimento é utilizado com inúmeras funções para buscar os dados no ROS, atualizar variáveis e atualização a interface de forma cicla, como mostrado na figura 6.1.

Listing 6.3: Posicionamento da imagem relativa a visão lateral.

```
<script>
setInterval(updateRearArmImage, 300);
setInterval(updateFrontArmImage, 300);
</script>
```

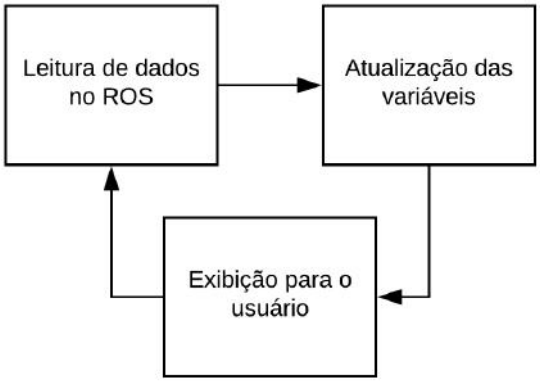

Figura 6.1: Fluxo para exibição de dados.

Todos os botões da interface estão associados a uma função que é passada como argumento na criação do botão no HTML, podemos ver um exemplo em 6.4. Nesse exemplo ao ser utilizar esse botão é chamada a função *onDefaultValuesStairTable*.

Listing 6.4: Criando botões na Interface.

```
<button class="button_default fixed_width"
 id='btSetStairClimbDefaultValues' onclick='onDefaultValuesStairTable()
    ,
 >Set Default Values</button>
```

6.3 Informações obtidas pelo ROS

No arquivo Ros.js foram colocadas todas as funções relativas ao ROS, sendo elas de conexão, leitura, escrita ou serviços. Dessa forma, todos os valores que são exibidos ou enviados para o robô passam pelas funções implementadas nesse arquivo.

A primeira implementação feita refere-se a conexão com o ROS, realizada conforme mostrado no capítulo 4. Utilizam-se as configurações do robô, além de uma função que é chamada quando o usuário clica para conectar-se, na figura 5.1, caso tenha perdido a conexão por algum motivo. Os erros são mostrados somente no console de JavaScript.

Muitas informações do robô são coletadas, tais como velocidade, posição dos braços, temperatura do computador, entre outros. Elas são mostradas na tela para que o usuário saiba como se encontra o robô e tenha melhores condições para teleoperação. Com isso, foram criadas funções que buscam essas informações em diversos tópicos e as salvam em variáveis, as quais são usadas pelas funções responsáveis pela exibição que serão faladas posteriormente.

A leitura, escrita em tópicos e a chamada de serviços foi explicada no capítulo 4. Nesse ponto, vamos nos concentrar no que cada função lê e o que é feito com os dados recebidos.

6.3.1 Posição e orientação do robô

A velocidade, posição e orientação do robô são publicadas no tópico */diane/diane_controller/state*, no qual é transmitida uma mensagem customizada do tipo *diane_controller/RobotState*. Nesta são enviadas duas listas, sendo elas *track_position*, que traz os valores das posições de cada esteira do robô, e *track_velocity*, que possui os valores da velocidade de cada esteira. É enviada uma mensagem do tipo pose que possui um quaternion, usado para calcular a orientação do robô, conforme mostrado a seguir.

O quaternion tem sua expressão mostrada na equação (6.1) e foi utilizado porque, em comparação com matrizes de rotação, é mais compacto e computacionalmente eficiente. No entanto, para a exibição, a expressão foi transformada para os ângulos de Euler *(roll , pitch e yaw)*, a fim de facilitar o entendimento. A fórmula usada para o cálculo é mostrada na equação (6.2).

$$q = q_o + iq_1 + jq_2 + kq_3 \tag{6.1}$$

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(2(q_0q_1 + q_2q_3)), 1 - 2(q_1^2 + q_2^2) \\ asin(2(q_0q_2 - q_3q_1)) \\ atan2(2(q_0q_3 + q_1q_2)), 1 - 2(q_2^2 + q_3^2) \end{bmatrix} \tag{6.2}$$

6.3.2 Coletando dados para o laser

As informações do radar são coletadas na função *readLaser* que busca os dados no tópico */r/scan* recebendo mensagens do tipo *sensor_msgs/LaserScan*, ao receber os

Tabela 6.1: Modos de controle do DIANE.

Modo de controle	Significado
0	Não controlado
1	Travado
2	Controle por corrente
3	Controle por velocidade
4	Controle por posição

valores é chamada a função *calculatePoints* para calcular os pontos a serem plotados, no qual o calculo será mostrado mais a frente.

6.3.3 Controle enviado para o robô

A função *actualInput* é responsável por buscar as informações sobre o controle que está sendo enviado para o robô, tanto o modo quanto os valores passados para cada esteira, e as transforma em medidas de velocidade para que possam ser exibidas. Essas informações são publicadas no tópico */diane/diane_controller/Motion/actual_input* a partir de mensagens do tipo *controller/Control*. Podemos ver o código dessa função em 6.5 no qual o significado dos modos de controle são mostrados na tabela 6.1.

O modo de controle define como verdadeira a variável da esteira ou braço, e o estado como não controlado, travado, desconectado ou controlado, não a diferença entre o modo de controle por velocidade ou posição, em ambos é considerado que esta sendo controlado e o componente será mostrado na cor verde.

Os dados transmitidos na variável data são os valores desejados de controle e, assim como o modo de controle, é enviada uma lista com três elementos que representam os valores da esteira, braço frontal e braço traseiro respectivamente. Esses valores são definidos em variáveis utilizadas para exibição na tabela de telemetria, sendo impossível que um elemento esteja em mais de um modo de controle ao mesmo tempo, deixando o módulo não utilizado com um hífen indicativo de que não há valor para aquele elemento.

Listing 6.5: Função actualInput.

```
function actualInput() {
        var listener = new ROSLIB.Topic({
        ros: ros,
        name: '/diane/diane_controller/Motion/actual_input',
        messageType: 'controller/Control'
});

listener.subscribe(function(message) {

        var encoder = new TextEncoder('utf8');
```

```
var modesdecoded = encoder.encode(atob(message.modes));

originId = message.originId;

if (modesdecoded[0] == 0) {
        tracks_not_controlled = true;
        tracks_controlled = false;
        tracks_disconnected = false;
        tracks_brake = false;
} else if (modesdecoded[0] == 1) {
        tracks_not_controlled = false;
        tracks_controlled = false;
        tracks_disconnected = false;
        tracks_brake = true;
} else if ((modesdecoded[0] == 3) || (modesdecoded[0] == 4)){
        tracks_not_controlled = false;
        tracks_controlled = true;
        tracks_disconnected = false;
        tracks_brake = false;
}

if (modesdecoded[1] == 0) {
        front_arm_not_controlled = true;
        front_arm_controlled = false;
        front_arm_disconnected = false;
        front_arm_brake = false;
} else if (modesdecoded[1] == 1) {
        front_arm_not_controlled = false;
        front_arm_controlled = false;
        front_arm_disconnected = false;
        front_arm_brake = true;
} else if ((modesdecoded[1] == 3) || (modesdecoded[1] == 4)){
        front_arm_not_controlled = false;
        front_arm_controlled = true;
        front_arm_disconnected = false;
        front_arm_brake = false;
}

if (modesdecoded[2] == 0) {
        rear_arm_not_controlled = true;
        rear_arm_controlled = false;
        rear_arm_disconnected = false;
        rear_arm_brake = false;
} else if (modesdecoded[2] == 1) {
        rear_arm_not_controlled = false;
        rear_arm_controlled = false;
        rear_arm_disconnected = false;
```

```
            rear_arm_brake = true;
        } else if ((modesdecoded[2] == 3) || (modesdecoded[2] == 4)) {
            rear_arm_not_controlled = false;
            rear_arm_controlled = true;
            rear_arm_disconnected = false;
            rear_arm_brake = false;
        }

        if (modesdecoded[0] == 3) { // data[0]: linear velocity  data
            [1]: angular velocity
            rightTrackVelocitySetPoint = (message.data[0] + 0.173*
                message.data[1]);    // R=0.173
            leftTrackVelocitySetPoint  = (message.data[0] - 0.173*
                message.data[1]);
            rightTrackPositionSetPoint = '-';
            leftTrackPositionSetPoint = '-';
        } else if (modesdecoded[0] == 4) {
            rightTrackPositionSetPoint = message.data[1];
            leftTrackPositionSetPoint = message.data[0];
            rightTrackVelocitySetPoint = '-';
            leftTrackVelocitySetPoint = '-';
        } else {
            rightTrackPositionSetPoint = '-';
            leftTrackPositionSetPoint = '-';
            rightTrackVelocitySetPoint = '-';
            leftTrackVelocitySetPoint = '-';
        }

        if (modesdecoded[1] == 3) {
            frontArmVelocitySetPoint = Math.round(message.data[2] *
                100) / 100;
            frontArmPositionSetPoint = '-';
        } else if (modesdecoded[1] == 4) {
            frontArmPositionSetPoint = Math.round(message.data[2] *
                100) / 100;
            frontArmVelocitySetPoint = '-';
        } else {
            frontArmVelocitySetPoint = '-';
            frontArmPositionSetPoint = '-';
        }

        if (modesdecoded[2] == 3) {
            rearArmVelocitySetPoint = Math.round(message.data[3] *
                100) / 100;
            rearArmPositionSetPoint = '-';
        } else if (modesdecoded[2] == 4) {
            rearArmPositionSetPoint = Math.round(message.data[3] *
```

```
                    100) / 100;
                rearArmVelocitySetPoint = '-';
        } else {
                rearArmVelocitySetPoint = '-';
                rearArmPositionSetPoint = '-';
        }

listener.unsubscribe();

});
}
```

Na mensagem de controle são transmitidos o id do emissor, *originId*, uma lista com os modos de controle, *modes*, e uma lista com o valor que deve ser aplicado, *data*. Uma observação importante é que tanto o id quanto os modos vêm no formato *uint8*, os quais são inteiros, com um menor valor máximo, utiliza menos *bytes*. Ao ler esse tipo de dado é necessário que ele seja codificado no formato de texto que se deseja ver. Utilizamos o padrão *UTF8*, vide a função mostrada em 6.6. Isso ocorre pois por padrão a decodificação é feita em base 64.

Listing 6.6: Como codificar uma mensagem que não venha em base 64.

```
        var encoder = new TextEncoder('utf8');
        var modesdecoded = encoder.encode(atob(message));
```

Não é possível saber quem está fazendo o controle somente pelo id. No entanto, recebemos um id quando requisitamos o controle, que será mostrado mais a frente no documento. Ao checarmos o id que está controlando podemos saber se é a interface ou outro dispositivo que está controlando o robô. Isso ocorre porque somente um dispositivo pode controlar o robô; as mensagens que chegam de outros id são rejeitas, até que o atual pare de enviar.

6.3.4 Informação de temperatura

A informação de temperatura do computador embarcado é publicada no tópico */diane/temperature* com uma mensagem do tipo *diagnostic_msgs/DiagnosticArray*. É enviada uma lista com valores de cada núcleo da CPU. Ao exibir para o usuário, mostra-se somente o núcleo com a maior temperatura.

6.3.5 Controle feito pela Interface

Existem algumas funções para que seja feito o controle do DIANE pela interface. A primeira delas é chamada de *requestID*, mostrada em 4.7 e 4.8, que requisita o serviço chamado */diane/diane_controller/request_id* do tipo *controller/RequestID* e recebe

como retorno o ID que será usado para fazer o controle. Essa função é solicitada quando o usuário demanda o controle pela primeira vez; nas próximas será usado o mesmo ID.

A função *verifyControl*, mostrada em 6.7, serve para verificar se o controle está sendo feito pela interface. Com ela, compara-se o id emissor dos comandos com o id indicado após a requisição e, em caso positivo, atualizam-se as variáveis da tabela de controle de modo que seja possível utiliza-la.

Listing 6.7: Função verifyControl usada para verificar se o robô está sendo controlado pela interface.

```
function verifyControl() {
        if (originId == control_id) {
                not_controlled = false;
                requesting_control = false;
                active_control = true;
                onbrake_control = false;
                robot_linear_velocity_SP = 0;
                robot_angular_velocity_SP = 0;
                received_control = true;
        }
}
```

Existem uma função para o controle de velocidade, *sendControlVelocity*, e outra para o de posição, *sendControlPosition*. Em ambas as funções o controle das esteiras é sempre dado por velocidade, a mudança é em relação ao controle dos braços. São passados quatro parâmetros para essas funções, que são: velocidade linear, velocidade angular, velocidade ou posição dos braços dianteiros e velocidade ou posição dos braços traseiros. Nestas funções é verificado se o ID já foi requisitado ou, caso contrário, ele a requisição é feita. A mensagem enviada é do tipo *controller/Control*. Podemos ver o no exemplo 6.8 a função de envio por velocidade.

Listing 6.8: Função sendControlVelocity usada para enviar o controle por velocidade ao robô.

```
function sendControlVelocity(linear_velocity, angular_velocity,
    fromt_arms, rear_arms) {
if (control_id == 0) {
        requestID();
        }
if (control_id != 0) {
        var controlMSG = new ROSLIB.Topic({
        ros: ros,
        name: '/diane/diane_controller/input',
        messageType: 'controller/Control'
        });
```

```
        var msg = new ROSLIB.Message({
                header: { stamp: {secs: date.getSeconds} },
        originId: control_id,
        modes: [3, 3, 3],
        data: [linear_velocity, angular_velocity, fromt_arms, rear_arms
            ]
        });

        controlMSG.publish(msg);
        }
}
```

Uma função que é chamada repetidamente para enviar o controle ao ROS é nomeada de *sendControl*. Ao desejar adicionar ou tirar algum modo de controle, é importante observar a lógica dessa função. Quando solicitada, ela verifica se o controle que está ativo é o interface, caso seja. envia a mensagem de controle com os parâmetros atuais, se não a mensagem enviada será com os valores de velocidade zero e com a posição atual do robô, para que ele fique parado quando o interface iniciar o controle. Caso esteja no modo *Break* é enviada uma mensagem para que o robô se mantenha nesse estado. Esse fluxo pode ser visto na figura 6.2 e o seu código em 6.9.

Listing 6.9: Função sendControl usada para chamar a função de controle desejada.

```
function sendControl() {
        if (!send_control) {
                return;
        }
        if (active_control) {
                if (onbrake_control) {
                        brakeControl();
                        return;
                }
                sendControlPosition(robot_linear_velocity_SP,
                    robot_angular_velocity_SP,
                    robot_front_arms_position_SP,
                    robot_rear_arms_position_SP);
        } else if (requesting_control) {
                sendControlVelocity(0, 0, 0, 0);
                verifyControl();
        }
}
```

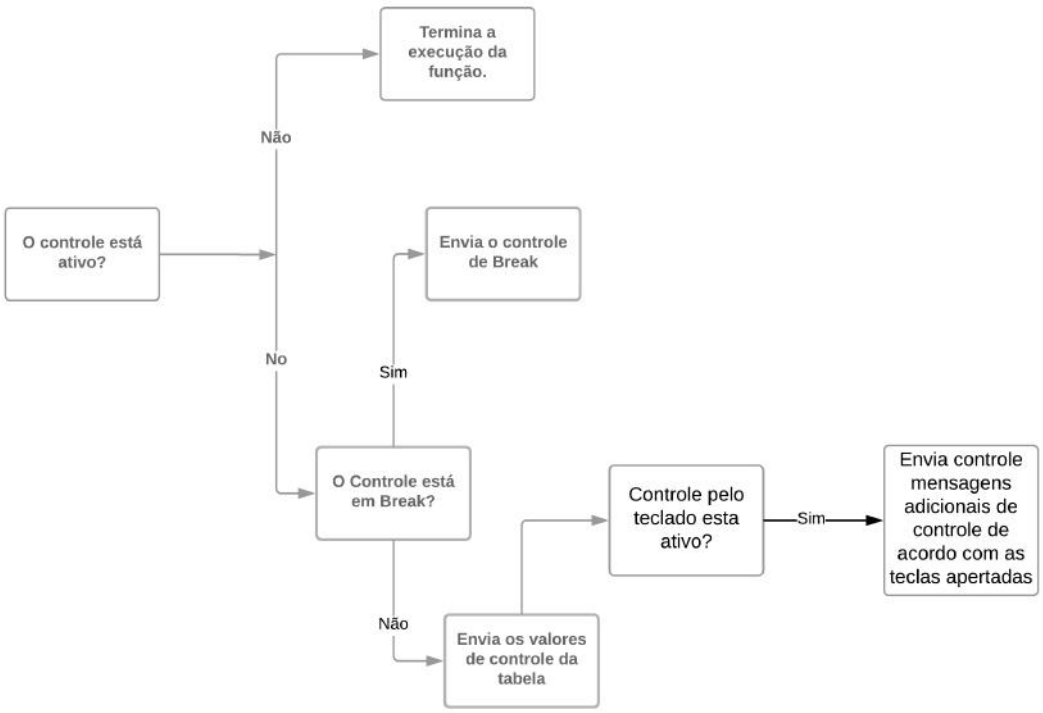

Figura 6.2: Fluxo para envio de controle pela interface.

6.3.6 Subida de escada

Para a subida de escada temos funções relacionadas tanto às etapas de aclive quanto para sua detecção.

Para identificar a escada é chamado um serviço de nome */diane/diane_octomap/-Detect_Stairs_From_Server* no qual não é necessário passar nenhum parâmetro, apenas deve-se posicionar o robô de frente para a escada, para facilitar que o algorítimo encontre um bom modelo. Como resposta ao serviço, recebemos os parâmetros calculados da escada, se encontrada. Essa função é pedida através do botão *detect Stair*.

Os outros três botões são definidos pelas funções *startClimb*, ativada pelo botão *Start Stair Climb*, *exitStair*, relacionada ao botão *Exit Stair*, e *endClimb*, relativa ao botão *Stop Stair Climb*. De caráter iniciador, a função *startClimb* precisar passar os parâmetros para a subida, os quais são valores que constam na tabela no momento em que o botão é apertado.

6.3.7 Botões de ativação e calibração

Para os botões de ativação foi feita uma função chamada *setDAOutput* que recebe como argumento o ID da porta, a qual deve ser ligada ou desligada. Dessa forma, toda vez que um botão é acionado, passa o ID correspondente ao botão o oposto do estado no qual ele se encontra no momento, ou seja, desliga o componente caso

Tabela 6.2: Variáveis do arquivo ControlTable.js

Variável	Significado
not_controlled	Define se a interface não está ativa
requesting_control	Define se está sendo requisitado o controle
active_control	Define se o controle está ativo
onbrake_control	Define se o controle está no modo *Break*
send_control	Define se deve-se enviar as mensagens de controle
received_control	Define se o id para enviar mensagens de controle já foi requisitado
control_id	Guarda o id recebido para o envio de mensagens

esteja ligado e o liga se estiver desligado. Dessa forma a EPOS liga ou desliga o relé correspondente.

A função *calibrateRobot* define a posição atual do robô como zero. O único requisito para que ela funcione é que a interface esteja controlando o robô.

6.4 Tabela de controle

No arquivo ControlTable.js estão todas as funções de manipulação da tabela de controle e do botão *break* relacionadas à exibição. Para tal, são usadas as variáveis definidas no início e que são atualizadas pelas funções presentes no arquivo Ros.js. Os significados de cada uma estão apresentados na tabela 6.2.

6.4.1 Gerando a tabela de Controle

A função *generateControlTable* é responsável pela criação da tabela e é chamada uma única vez. Caso seja necessário adicionar campos ou alterar o estilo da tabela, essa função deve ser modificada.

6.4.2 Funcionamento

Enquanto não for requisitado o controle pela interface, os dados da tabela serão atualizados de acordo com os dados que estão sendo enviados pelo robô. Isso é importante, pois ao requisitar o controle começarão a ser transmitidas mensagens de controle periodicamente. Para garantir que o robô iniciará parado, é preciso que as posições dos braços frontais e traseiros estejam de acordo com a atual. Além disso, no momento que for chamado, deve-se estabelecer velocidades linear e angular iguais à zero.

Com o controle ativo, a tabela termina no último estado e para de ser atualizada a fim de que o usuário consiga escolher os seus parâmetros. Estes desvem estar dentro de intervalo para impedir que sejam passados números que claramente estão errados, ou mensagens que não são numéricas. Existe uma função para cada célula

que verifica o valor do parâmetro e gera um alerta caso esse valor não esteja dentro do permitido.

Entretanto, em nem todos os momentos é permitido que o operador atualize as células. Na verdade, elas somente podem ser alteradas quando estamos no estado de controle. Para garantir isso, existe uma função chamada *updateCellEditability* que bloqueia ou libera a edição das células de acordo com o estado atual.

Assim como a tabela, os botões também possuem uma função, a *updateControlButtons*, com o intuito de deixá-los disponíveis ou não para o usuário, conforme o estado atual. A exemplo disso, o botão de alterar os parâmetros que estão sendo enviados só se encontrará habilitado quando o controle estiver ativo.

Os estados na tabela de controle são indicados pelo semáforo. A função *updateRobotControlStateLight* observa qual o estado atual e o atualiza, ligando a luz correspondente de acordo com a tabela 5.2.6.

Ainda existem as funções que são chamadas para cada botão disponível e tem seus estados alterados de acordo com eles.

6.5 Controle utilizando o teclado

Um dos modos de controle possível é através do teclado, no qual os comandos foram mostrados na tabela 5.2. O código que torna isso possível está todo no arquivo *keyboardteleop.min*. Seu funcionamento consiste em esperar os eventos proveniente do teclado e, quando ele se dá, executar a ação de acordo com as teclas pressionadas. É necessário que o controle pelo teclado esteja ativo, enviando um controle de velocidade ao final da execução.

6.6 Visão frontal e lateral do robô

No arquivo RobotViews.js temos as variáveis e funções relacionadas às visões frontais e laterais do robô, mostradas na parte superior da interface. As variáveis usadas nessa parte tem relação com o tipo de controle que está sendo utilizado no momento, a fim de identificá-lo para o usuário através da cor e do ângulo dos braços. A explicação destas consta na tabela 6.3.

Todas as funções nesta parte do código servem para atualizar as cores das imagens de acordo com o estado atual, que estão descritas na tabela 5.2.1, e a posição dos braços de acordo com seu ângulo. Essas funções são chamadas periodicamente, atualizando as imagens.

Tabela 6.3: Variaveis do arquivo RobotViews.js

Variável	Significado
tracks_not_controlled	Mostra se as esteiras não estão sendo controladas
tracks_controlled	Mostra se as esteiras estão sendo controladas
tracks_disconnected	Mostra se as esteiras estão desconectadas
tracks_brake	Mostra se as esteiras estão no estado de *Break*
front_arm_not_controlled	Mostra se o braço frontal não está sendo controlado
front_arm_controlled	Mostra se o braço frontal está sendo controlado
front_arm_disconnected	Mostra se o braço frontal está desconectado
front_arm_brake	Mostra se o braço traseiro está em estado de break
rear_arm_not_controlled	Mostra se o braço traseiro não está sendo controlado
rear_arm_controlled	Mostra se o braço traseiro está sendo controlado
rear_arm_disconnected	Mostra se o braço traseiro está desconectado
rear_arm_brake	Mostra se o braço traseiro está em estado de break
front_arm_angle	Guarda o valor do ângulo do braço frontal
front_arm_angle_offset	Guarda o valor do *offset* do braço frontal usado na calibração
rear_arm_angle	Guarda o valor do ângulo do braço traseiro
rear_arm_angle_offset	Guarda o valor do *offset* do braço traseiro usado na calibração

6.7 Laser

O arquivo Laser.js contem as funções utilizadas para criação e atualização do radar. A função que desenha o laser, *createLaser*, é chamada para fazer as marcações do laser, sem a plotagem dos pontos.

A função *calculatePoints* é responsável por calcular as coordenadas do pontos no plano cartesiano. Para entendermos como é feito esse calculo precisamos primeiramente analisar de que forma recebemos os dados. As mensagens do tipo *laserScan* enviam os dados da menor e maior medição de ângulo possível pelo dispositivo, no caso do hokuyo de $-135°$ a $135°$. O primeiro valor corresponde ao menor ângulo e o último ao maior do array enviado com as distâncias, sendo todos os pontos igualmente espaçados. Assim podemos calcular os valores de x e y usando as relações trigonométricas mostradas na equações (6.3) e (6.4) obtidas a partir da figura 6.3.

$$x = Distancia \times \cos(\alpha) \tag{6.3}$$

$$y = Distancia \times \sin(\alpha) \tag{6.4}$$

A função *updateRadar* recebe os pontos x e y calculados e os exibe na tela. Primeiramente é chamado a função *clearRadar* que remove os pontos calculados anteriormente e então são colocados os novos pontos no laser. Os parâmetros utilizados por essa função que estão na tabela correspondente ao laser e foram explicado no capítulo 5. É feito uma proporção para se calcular os pontos de acordo com o

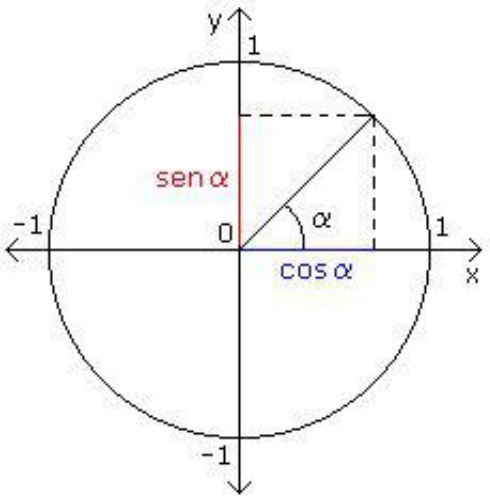

Figura 6.3: Fluxo para exibição de dados.

tamanho máximo que deve ser exibido e o step é utilizado para saber de quantos em quantos pontos recebidos do laser deve-se exibir no radar.

Listing 6.10: Função updateLaser usada para os pontos no laser.

```
function updateLaser(points) {
        var c = document.getElementById("radar");
        var ctx = c.getContext("2d");
        clearLaser();
        createLaser();
        for(i = 0 ; i < points.length ; i = i+stepLaser){
                ctx.beginPath();
                ctx.fillRect(points[i][0]*150/maxLaserRange +150 ,
                        points[i][1]*150/maxLaserRange + 150 ,1,1);

                ctx.stroke();
        }
}
```

6.8 Tabela de telemetria

O arquivo TelemetryTable.js possui as variáveis, que são utilizadas para mostrar os valores na tabela e estão descritas na tabela 6.4 de telemetria, as funções de criação e atualização da tabela. A função de criação é chamada uma única vez; a função de atualização, *updateTelemetryTable*, é chamada repetidas vezes para que se retifiquem os valores.

Tabela 6.4: Variáveis do arquivo TelemetryTable.js

Variável	Significado
leftTrackPositionSetPoint	Guarda o valor da posição desejada para esteira esquerda
leftTrackPosition	Guarda o valor da posição da esteira esquerda
leftTrackVelocitySetPoint	Guarda o valor da velocidade desejada para esteira esquerda
leftTrackVelocity	Guarda o valor da velocidade da esteira esquerda
rightTrackPositionSetPoint	Guarda o valor da posição desejada para esteira direita
rightTrackPosition	Guarda o valor da posição da esteira direita
rightTrackVelocitySetPoint	Guarda o valor da velocidade desejada para esteira direita
rightTrackVelocity	Guarda o valor da velocidade da esteira direita
frontArmPositionSetPoint	Guarda o valor da posição desejada para os braços frontais
frontArmPosition	Guarda o valor da posição dos braços frontais
frontArmVelocitySetPoint	Guarda o valor da velocidade desejada para os braços frontais
frontArmVelocity	Guarda o valor da velocidade dos braços frontais
rearArmPositionSetPoint	Guarda o valor do ângulo do braço frontal
rearArmPosition	Guarda o valor do *offset* do braço frontal usado na calibração
rearArmVelocitySetPoint	Guarda o valor do ângulo do braço traseiro
rearArmVelocity	Guarda o valor do *offset* do braço traseiro usado na calibração

Tabela 6.5: Variáveis do arquivo PhysicalInfo.js

Variável	Significado
battery_charge	O valor da carga da bateria
battery_voltage	O valor da tensão da bateria
battery_current	O valor da corrente da bateria
CPU_Temperature	O valor da temperatura do core mais quente da CPU
DIANE_Pitch	O valor do *pitch* obtido pelo kinect

6.9 Informações físicas

O arquivo PhysicalInfo.js traz as informações físicas do robô, como a carga da bateria, temperatura do núcleo mais quente da CPU e o *pitch* do robô, passado pelo kinect. Assim como outras funções de exibição, essas também são chamadas periodicamente para atualização dos valores. As variáveis aqui definidas são detalhadas na tabela 6.5.

6.10 Conexão com o ROS

As imagens de conexão que se encontram no cabeçalho da página são exibidas caso a interface tenha conseguido ou não se conectar ao ROS. A função *updateConnectionImage* define a imagem a ser exibida e está no arquivo ConnectionState.js.

Tabela 6.6: Variáveis do arquivo StairClimb.js

Variável	Significado
stair_Number_Steps	Guarda o valor do número de degraus da escada
stair_Angle	Guarda o valor do ângulo da escada
stair_Step_Length	Guarda o valor do comprimento da escada
stair_Step_Width	Guarda o valor da largura da escada
stair_Step_Height	Guarda o valor da altura da escada
stair_climb_not_controlled	Define se o controle de subida de escada não está ativo
stair_climb_paused_control	Define se o controle de subida de escada está pausado
stair_climb_active_control	Define se o controle de subida de escada está ativo

6.11 Tabela de subida de escada

As variáveis e funções relacionadas à subida de escada encontram-se no arquivo StairClimb.js. As variáveis que aqui se apresentam estão relacionadas aos cinco valores presentes na tabela, correspondentes ao número de degraus, ângulo, comprimento, largura e altura da escada. Aquelas relacionadas ao estado do controle de subida são mostradas juntas às suas definições na tabela 6.6.

Como nas outras tabelas, existe uma função para a criação, *generateStairTable*, e atualização, *updateStairTable* dos valores. A função de criação da tabela é chamada somente uma vez no início do programa e a de atualização é chamada periodicamente.

Também possui a função *onDefaultValuesStairTable*, relacionada ao botão *Set Default Values*, o qual define como valor padrão 25° para o ângulo da escada e 130 para a altura.

Capítulo 7

Conclusões e Trabalhos Futuros

Este trabalho apresentou uma interface web para o robô DIANE, tendo como intuito facilitar a navegação, diminuir o tempo de treinamento do operador e deixar a navegação mais segura e eficiente, diminuindo as chances de acidentes e danos ao robô.

A interface web também resolve o problema do ROS em relação à acessibilidade. Com essa aplicação é possível acessar e controlar o robô de praticamente qualquer dispositivo e, caso exista conexão com a internet, de outros locais.

A comunicação da interface foi feita utilizando a biblioteca *roslibjs*. Está possibilitou a leitura de tópicos, que é importante para a aquisição de dados que são exibidos, a escrita em tópicos que permitiu o controle do robô. A interface foi desenvolvida utilizando HTML e CSS para exibição e JavaScript para dar dinâmica a página, atualizando os valores de tabelas, cores das imagens entre outras funções.

Outro fator interessante mostrado é que a implementação da interface web não está diretamente ligada ao formato do sistema utilizado no software embarcado do robô. Dessa forma, o número de possibilidades de exibição e tratamento dos dados sensoriais aumenta, permitindo uma dinâmica de operação mais versátil e completa.

7.1 Trabalhos Futuros

Com a interface web apresentada, é possível propôr melhorias que agregariam novas possibilidades para o operador, tais como:

- Colocar a câmera do robô na página. Essa não faz parte do sistema ROS, não sendo um tópico que passará os parâmetros da câmera. No entanto, por ser uma página baseada em *JavaScript*, é possível ler a porta da câmera e coloca-lá na página.

- Adicionar outras opções de controle. Poderiam ser incorporados outros modos de controle, como através da posição das esteiras ou da velocidade dos braços. Para tal, não teria que ser rigorosamente necessário fazer uso das tabelas nas quais são inscritos os valores. Além disso, deveria ser possível utilizar botões para o controle e habilitá-lo pelo teclado.

- Adicionar a nuvem de pontos obtida pelo Laser/*Kinect* para obter-se a ideia de como está sendo feito o mapeamento pelo robô, além de mostrar a escada que foi modelada, a fim de saber se foi encontrado um bom modelo.

- Utilização de um framework de Front-End que utilize JavaScript como React, Angular, Vue, etc.

- Possibilitar o controle utilizando o Joystick através da página.

Referências Bibliográficas

LEE, J. "Web Applications for Robots using rosbridge", *Brown University*, 2012.

ALEXANDER, B., HSIAO, K., JENKINS, C., et al. "Robot web tools [ros topics]", *IEEE Robotics & Automation Magazine*, v. 19, n. 4, pp. 20–23, 2012.

OSENTOSKI, S., JAY, G., CRICK, C., et al. "Robots as web services: Reproducible experimentation and application development using rosjs". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 6078–6083. IEEE, 2011.

TORIS, R., KAMMERL, J., LU, D., et al. "Robot Web Tools: Efficient messaging for cloud robotics. 2015 IEEE". In: *RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

CRICK, C., OSENTOSKI, S., JAY, G., et al. "Human and robot perception in large-scale learning from demonstration". In: *Proceedings of the 6th international conference on Human-robot interaction*, pp. 339–346. ACM, 2011.

CHUNG, M. J.-Y., FORBES, M., CAKMAK, M., et al. "Accelerating imitation learning through crowdsourcing". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 4777–4784. IEEE, 2014.

LOPEZ, D., CEDAZO, R., SÁNCHEZ, F. M., et al. "Ciclope robot: Web-based system to remote program an embedded real-time system", *IEEE Transactions on Industrial Electronics*, v. 56, n. 12, pp. 4791–4797, 2009.

QUIGLEY, M., CONLEY, K., GERKEY, B., et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*, v. 3, p. 5. Kobe, 2009.

AZEVEDO, T. P. *Locomoção de um robô móvel com esteiras em escadas*. Graduate dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, 2017.

FOUNDATION, O. S. R. "ROS Wiki". 2018. Disponível em: <http://wiki.ros.org/>.

CHAN, D. K. S. *Detecção e Modelagem Eficiente de Escadas para Controle de Subida de robôs móveis*. Graduate dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, 2017.

LOUREIRO, G. S. M. *Desenvolvimentode software para posiconamento dinâmico do ROV LUMA*. Graduate dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, 2017.

DE LIMA, L. C. "Sensing system design and torque analysis of a haptic operated climbing robot". UFRJ, 2016.

Apêndice A

Launch para inicialização do robô

Segue o código dos *launch* usados para inicializar o robô:

Listing A.1: Arquivo robot.launch

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
<arg name="bag_location" default="" />
<arg name="config_location" default="$(find diane_files)/config/" />
<arg name="controller_name" default="diane_controller"/>
<arg name="manager" default="diane_manager"/>
<arg name="max_velocity" default="0.25"/>
<arg name="max_idle_speed" default="0.02"/>
<arg name="namespace" default="r"/>

<node ns="$(arg namespace)" pkg="nodelet" type="nodelet" name="$(arg
    manager)" args="manager"/>
<node ns="$(arg namespace)" pkg="nodelet" type="nodelet" name="epos1"
    args="load epos/EposNodelet $(arg manager) _max_position_error:=2
    _personal_id:=101 _can_device:=can0 _device_id:=1 _inverted:=false
    _velocity:=0.3 _acceleration:=2000 _deceleration:=2000
    _gain_position:=920000 _gain_velocity:=28200">
<!--Epos esquerda corresponde ao id 1-->
</node>
<node ns="$(arg namespace)" pkg="nodelet" type="nodelet" name="epos2"
    args="load epos/EposNodelet $(arg manager) _max_position_error:=2
    _personal_id:=102 _can_device:=can0 _device_id:=2 _inverted:=true
    _velocity:=0.3 _acceleration:=2000 _deceleration:=2000
    _gain_position:=920000 _gain_velocity:=28200">
<!--Epos direita corresponde ao id 2-->
</node>
<node ns="$(arg namespace)" pkg="nodelet" type="nodelet" name="epos3"
    args="load epos/EposNodelet $(arg manager) _max_position_error:=270
     _personal_id:=103 _can_device:=can0 _device_id:=8 _inverted:=false
     _acceleration:=2000 _deceleration:=2000 _gain_position:=37000
    _gain_velocity:=1095">
```

```
<!--Epos dianteira corresponde ao id 8-->
</node>
<node ns="$(arg namespace)" pkg="nodelet" type="nodelet" name="epos4"
    args="load epos/EposNodelet $(arg manager) _max_position_error:=270
     _personal_id:=104 _can_device:=can0 _device_id:=4 _inverted:=false
     _acceleration:=2000 _deceleration:=2000 _gain_position:=37000
    _gain_velocity:=1095">
<!--Epos traseira corresponde ao id 4-->
<rosparam param="digital_outputs">[1,2,3]</rosparam>
<rosparam param="analog_inputs">[0,1]</rosparam>

</node>

<node ns="$(arg namespace)" pkg="nodelet" type="nodelet" name="$(arg
    controller_name)" args="load diane_controller/
    DianeControllerNodelet $(arg manager) _name:=Motion _max_velocity:=
    $(arg max_velocity) _config:=$(arg config_location)controller
    _max_idle_speed:=$(arg max_idle_speed)" />
<node ns="$(arg namespace)" pkg="nodelet" type="nodelet" name="
    diane_remap" args="load diane_controller/DianeControllerRemap $(arg
     manager) _controller_name:=$(arg controller_name)" />
<!--node pkg="urg_node" type="urg_node" name="laser_horizontal" args="
    load urg_node diane_manager _ip_address:=172.16.0.10" />-->
<!--<include file="/opt/ros/indigo/share/freenect_launch/launch/
    freenect.launch" />-->

</launch>
```

Listing A.2: Arquivo laser.launch

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
<arg name="ip_address" default="172.16.0.10"/>
<arg name="namespace" default="r"/>
<arg name="manager" default=""/>

<node ns="$(arg namespace)" pkg="urg_node" type="urg_node" name="
    urg_node" args="load urg_node $(arg manager) _ip_address
    :=172.16.0.10 "/>
</launch>
```