Validação do Conceito Gêmeo Digital na Robótica Móvel

Gustavo Henrique Del Conte
gustavohdelconte@gmail.com
Laboratório ROSIE
Centro Universitário UniCuritiba
Curitiba, PR, Brasil

Jean Feliphe dos Santos
jeanliphe.kzn@gmail.com
Laboratório ROSIE
Centro Universitário UniCuritiba
Curitiba, PR, Brasil

Carlos Eduardo Magrin
carlos.magrin@unicuritiba.com.br
Laboratório ROSIE
Centro Universitário UniCuritiba
Curitiba, PR, Brasil

ABSTRACT

Inspired by the article "Creating a Digital Twin as an Open Source Learning Tool for Mobile Robotics" [17], this project aims to validate the concept of Digital Twin by virtualizing the Tupy mobile robot platform, developed by the ROSIE group at the University of Curitiba (UniCuritiba), for educational purposes. The goal is to explore cyber-physical systems and apply Industry 4.0 concepts, using the Robot Operating System (ROS) as the central system. The Tupy robot integrates with the Raspberry Pi (RPi) and Arduino Uno platforms through the ROS Melodic framework, allowing it to collect information from the mobile robot and control it remotely. Thus, successfully implementing the Digital Twin of Mobile Robot (DTMR) enables information exchange between different development platforms and virtualization in the CoppeliaSim - EDU simulator. The files for this project are available on http://github.com/rosie-projects/dtmr_tupy.

KEYWORDS

Sistemas Ciber-Físicos; Gêmeos Digitais; Robótica Móvel; Indústria 4.0; ROS.

1 INTRODUÇÃO

A robótica móvel é uma área em constante evolução. Os robôs móveis, uma vez restritos a grandes sistemas computacionais, agora podem ser construídos de forma compacta e eficiente, graças ao desenvolvimento de dispositivos de baixo custo e computadores embarcados [3]. Essa transformação possibilita a criação de robôs mais acessíveis e versáteis, capazes de realizar uma ampla gama de tarefas em diferentes ambientes.

Ao combinar os princípios da robótica móvel com as vantagens oferecidas pelos gêmeos digitais, tem-se uma abordagem promissora para o desenvolvimento e aprimoramento desses sistemas. Com isso, é possível simular e validar o comportamento do robô em um ambiente virtual antes mesmo de sua construção física [21].

Portanto, ao longo deste artigo, examinaremos os fundamentos teóricos e práticos relacionados ao conceito de gêmeo digital [17], bem como os métodos e tecnologias necessários para sua implementação. Além disso, discutiremos os resultados de nossa pesquisa e as possíveis aplicações do gêmeo digital do robô móvel Tupy como uma ferramenta educacional na área de robótica móvel.

2 FUNDAMENTAÇÃO TEÓRICA

Para desenvolver um robô móvel dentro do conceito de gêmeo digital, é essencial realizar uma pesquisa abrangente em duas áreas-chave: robótica móvel e indústria 4.0. Na robótica móvel, é importante compreender conceitos de mecânica, elétrica, eletrônica, sistemas operacionais como o Linux e o *framework* ROS (*Robot Operating System* - sistema operacional de robôs), além de tópicos relacionados à computação. Em relação à indústria 4.0, é fundamental entender os Sistemas Ciber-Físicos e os conceitos de Gêmeo Digital.

2.1 Industria 4.0

O termo Indústria 4.0 originou-se em 2011 na Alemanha, impulsionado pelos avanços tecnológicos na produção industrial e da tecnologia da informação emergentes no período, considerado assim como o início de uma quarta revolução industrial [8, 13]. Segundo autores, a indústria 4.0 é um conjunto de tecnologias alavancadas pela internet que visam integrar processos, pessoas, máquinas, objetos e produção para formar a chamada *fábrica inteligente*. A quarta revolução, traz todo esse contexto para o ambiente digital e descentraliza os serviços, através dos CPS, IoT (*Internet of Things* - internet das coisas) e IoS (*Internet of Services* - internet de serviços) [11, 21].

2.2 Sistemas Cíber-Físicos

Os CPS são uma nova geração de tecnologias que englobam outros conceitos bem conhecidos, como a IoT, indústria 4.0, M2M (*Machine to Machine* - comunicação entre máquinas), IoS e cidades inteligentes [9, 15], tecnologias presentes em nosso cotidiano, são aplicadas em diversas áreas, como: saúde, transporte, cibersegurança, controle de infraestrutura crítica, eletrônica, sistemas de tráfego, manufatura e robótica distribuída [1, 19].

Entretanto, ainda há barreiras que precisam ser ultrapassadas, como: a criação de uma arquitetura sólida que suporte lidar com um número enorme de dispositivos conectados, estes de diferentes complexidades; um sistema de autoproteção contra-ataques exteriores e que garanta a segurança dos dados; uma grande capacidade de armazenamento de dados de maneira distribuída entre os dispositivos; uma conexão de alta velocidade e em tempo real [9]. Podemos considerar os CPS como um agente chave para o desenvolvimento tecnológicos que vai mudar a forma vamos interagir com o mundo físico [2, 12].

Nas arquiteturas de Sistemas Ciber-Físicos (CPS) as aplicações são compostas por duas camadas principais de tecnologia: a operacional (física) e a da informação (*cyber*). Os protocolos de comunicação nessas arquiteturas podem variar entre protocolos industriais e tecnologia da informação. A integração entre as camadas física e cibernética ocorre por sensores e atuadores. Os sensores são dispositivos que captam informações do ambiente físico e as convertem em sinais elétricos e digitais, sendo então utilizados pelos sistemas de monitoramento e controle em tempo real. Por outro lado, atuadores são responsáveis por controlar diversas características do ambiente físico, com base em sinais e comandos digitais, como acionamento de válvulas, resistências, motores, entre outras [21].

2.3 Robótica Móvel

A Robótica é uma área interdisciplinar que combina conhecimentos de diversas disciplinas, incluindo engenharia mecânica, engenharia elétrica, ciência da computação, inteligência artificial, entre outras. Ela se originou da teoria de controle, cibernética e inteligência artificial e busca desenvolver sistemas autônomos capazes de interagir com o ambiente e realizar tarefas específicas de forma eficiente e autônoma. O conhecimento nessas áreas é fundamental para o avanço e desenvolvimento da robótica [23, 24].

Os robôs móveis podem se deslocar de forma inteligente em ambientes livres por meio de um processo de controle contínuo, captando e interpretando dados do ambiente para se localizar e planejar suas ações [22]. Sua arquitetura versátil permite uma ampla gama de aplicações, desde reconhecimento e inspeções até intervenções e assistência em diferentes ambientes, como exploração planetária e operações militares [4, 6].

2.4 Sistemas Operacionais

Um Sistema Operacional (SO) é um *software* ou conjunto de *softwares* responsável pela integração entre um sistema microprocessado e o usuário [7]. Geralmente, esses sistemas oferecem funções de gerenciamento de tarefas, memória e recursos.

Os sistemas operacionais são amplamente utilizados em uma variedade de dispositivos. Nos computadores, exemplos populares incluem o Windows, Linux e MacOS, enquanto nos dispositivos móveis, como *smartphones* e *tablets*, o Android e o iPadOS são os mais comuns.

Em sistemas embarcados, a implementação de sistemas dedicados vem se tornando cada vez mais comum. No entanto, em contraste com os sistemas operacionais multitarefas, é crucial que os sistemas embarcados respeitem a premissa do processamento em tempo real. Ademais, em contextos específicos, como na robótica, é comum a utilização de sistemas voltados para o gerenciamento e integração de serviços e conexões, como o *Robot Operating System* (ROS).

2.4.1 Sistemas Operacionais Linux. O Linux é um sistema operacional amplamente utilizado, conhecido por sua confiabilidade e flexibilidade, além de ser de código aberto [16]. Com diversas distribuições disponíveis, como Debian e Ubuntu, o Linux é utilizado em uma variedade de dispositivos, desde *smartphones* até servidores. Na robótica móvel, o Linux é comum, especialmente em microcomputadores como o Raspberry Pi, permitindo que o robô atue como um servidor acessível remotamente [16]. Além disso, em dispositivos embarcados, os RTOS (*Real Time Operating System* - sistema operacional de tempo real) desempenham um papel importante, garantindo desempenho confiável em tarefas específicas [7]. Ao combinar as vantagens do Linux com os sistemas RTOS, é possível alcançar um equilíbrio ideal entre flexibilidade e desempenho em diversas aplicações.

2.4.2 Sistemas Operacionais Robóticos. Similar aos meta sistemas operacionais, o ROS é uma plataforma de código aberto projetada para simplificar o desenvolvimento de aplicativos de robótica, especialmente para robôs móveis. Ele oferece uma arquitetura flexível que permite o gerenciamento e a integração eficientes de hardware e software, possibilitando a criação de sistemas robóticos avançados. Além disso, o ROS estabelece uma rede na qual todos os processos

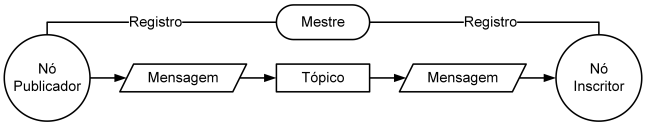

Figure 1: A Figura mostra o conceito da arquitetura, *framework*, ROS. O Mestre coordena a comunicação entre os nós, enquanto um tópico é um canal de comunicação usado para a troca de mensagens entre os nós, permitindo a transmissão e recebimento assíncronos de dados, com o registro mantendo um controle das conexões e coordenando a troca de mensagens. Fonte: Adaptado de ROS [20]

estão interligados, permitindo a comunicação e o compartilhamento de dados entre os nós do sistema, o que é essencial para a colaboração e o funcionamento eficaz de sistemas robóticos complexos [18]. A seguir, destaca-se alguns termos fundamentais do ROS que são essenciais para uma compreensão aprofundada para avançarmos com o projeto. A Figura 1 mostra o conceito da arquitetura do ROS [20].

Mestre (*Master*) fornece registro de nome DNS e pesquisa para o resto dos nós, sem ele não haverá comunicação entre os nós, serviços, mensagens e outros.

Nós (*Nodes*) são processos que executam tarefas computacionais e podem ser escritos em Python com *rospy* ou em C++ com *roscpp*. Por exemplo, a biblioteca OpenCV é utilizada para processar dados de sensores, como visão computacional em tempo real.

Tópicos (*Topics*) é a maneira que o ROS encontra para realizar o roteamento das mensagens pela rede. Quando um Nó está enviando dados, diz que o nó está publicando um tópico com a função *Publisher*. Para os nós receberem tópicos de outros nós, simplesmente escreve no tópico de interesse com a função *Subscriber*. Exemplo tópico */cmd_vel*, responsável pelo movimento do robô.

Mensagens (*Messages*) são informações enviadas através conexão entre os nós. Existem diversos tipos de mensagens padrões, por exemplo, *geometry_msgs/msg/Twist.msg*.

Serviços (*services*) possibilita interação com nós, por um arquivo que descreve o servidor, quais os tipos de dados que irão trafegar na requisição e qual tipo de dado é devolvido na resposta.

O ROS é baseado em vários princípios e compreende um número crescente de bibliotecas de código aberto. Dessa forma, existem outros conceitos e ferramentas completares importantes [3], sendo:

Visualizador (*rviz*) exibe graficamente a posição e a orientação de um robô em seu ambiente, com uma representação visual dos dados de seus sensores, por exemplo.

Ferramentas de Interface (*rqt*) pode ser usado para desenvolver interfaces de usuário específicas do ROS, por exemplo, utilização da ferramenta *rqt_graph* para visualizar o gráfico de computação do ROS.

Gazebo é uma ferramenta de simulação de código aberto amplamente usada em robótica e engenharia, oferecendo um ambiente 3D para modelagem, simulação e visualização de robôs, sensores e ambientes. O Gazebo é integrado ao ROS e permite testar algoritmos de controle e percepção em um ambiente virtual antes mesmo da implementação em *hardware*. Possui recursos avançados, como

física realista e simulação de sensores, e é altamente extensível através de *plugins*.

CoppeliaSim é um *software* de simulação de robótica 3D utilizado em pesquisa, desenvolvimento e educação. O simulador oferece uma interface gráfica intuitiva para modelagem, simulação e controle de robôs em ambientes complexos e dinâmicos. O CoppeliaSim suporta uma variedade de linguagens de programação, incluindo C/C++, Python, Lua e Matlab, permitindo uma integração flexível com diferentes sistemas e aplicativos.

Modelagem URDF ou Formato de Descrição de Robô Unificado (*Unified Robot Description Format*), é um formato de arquivo XML usado no ROS para descrever modelos de robôs. Ele especifica a geometria, a cinemática, a inércia e outras propriedades físicas dos *links* e juntas de um robô, permitindo a visualização e a simulação precisa de sua estrutura e comportamento.

3 DESENVOLVIMENTO

Durante o desenvolvimento do sistema físico, a fabricação de peças foi otimizada com a utilização da manufatura aditiva por meio da impressão 3D, complementada pela usinagem de perfis em alumínio. Na esfera eletrônica, a eletrônica embarcada foi fundamental na aquisição de sinais dos sensores ultrassônicos e controle dos atuadores, viabilizando um controle dos motores. A integração entre os periféricos (Kinect, *Desktop*, Arduino e RPi) foi viabilizada pela adoção de diversos protocolos de comunicação, enquanto a integração dos sistemas virtuais com os sistema operacional Linux e *framework* robóticos (ROS) foi essencial para o sucesso do projeto. A interação dos periféricos foi estabelecida por meio de redes (Wi-Fi), protocolos de comunicação (USB), linguagens de programação (Lua Script, C++, Python), *framework* (ROS Melodic) e virtualização (CoppeliaSIM), garantindo uma integração fluida e eficiente.

A integração do robô Tupy foi viabilizada por meio dos conceitos de Sistemas Ciber-Físicos (CPS), os quais possibilitaram a interligação os elementos computacionais com o ambiente físico. Isso permite não apenas o monitoramento e controle de periféricos em tempo real, mas também a realização de testes e simulações de processos físicos a partir do ambiente virtual. Esse referencial teórico fundamentou a criação da tríade (Figura 2) para conceber um Gêmeo Digital de um Robô Móvel.

Com base nos conceitos de Sistemas Ciber-Físicos (CPS) e na pesquisa apresentada no artigo [17], foi possível desenvolver um gêmeo digital para um robô móvel. Esse avanço tecnológico foi alcançado utilizando o *framework* ROS como uma plataforma de integração. A Figura 3 mostra o resultado prático desse processo, destacando a aplicação efetiva dos princípios teóricos na construção do gêmeo digital.

3.1 Sistema Físico

O sistema físico do DTMR Tupy é constituído pela adequação da plataforma, construção, modelagem, e montagem do modelo virtual do robô. A Figura 4 mostra a implementação e integração dos periféricos no sistema físico do robô [5].

3.1.1 Unidade Mecânica. A estrutura mecânica do DTMR Tupy, usou como base para a modelagem 3D, conceitos mecânicos com sistema computacional, sendo assim foi possível validar o conceito do design estrutural do DTMR Tupy e validar algumas melhorias

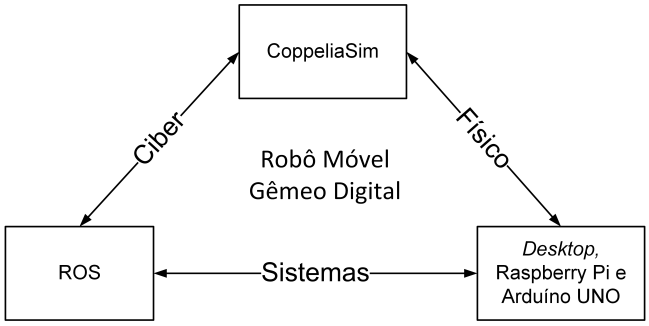

Figure 2: Tríade do DTMR baseada na arquitetura CPS [21], onde o CoppeliaSim representa a "computação", os periféricos representam o "controle" e o ROS representa a "comunicação", resultando em um Sistema Ciber Físico que proporciona um Gêmeo Digital de um robô móvel.

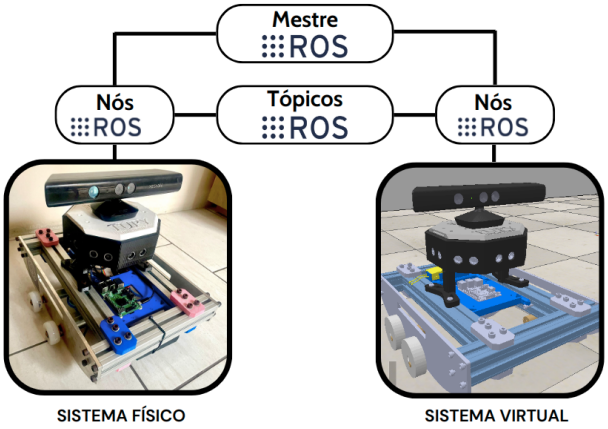

Figure 3: A estrutura DTMR Tupy com o framework ROS. O Mestre é o dispositivo central que controla a comunicação, os nós são dispositivos individuais em uma rede e os tópicos ROS são canais de comunicação assíncrona para trocar mensagens entre as partes de um sistema robótico.

em relação a sua primeira versão [17]. Melhorias estruturais para a redução de massa estrutural, e pontos de referência para a automação foram possíveis através da validação do conceito DTMR [10, 17].

Na avaliação do *chassi* central, composto por perfis estruturais de alumínio, é possível encontrar em sua estrutura uma flexibilidade para montagem de componentes externos e acessórios no seu chassi. A Figura 5 mostra o *chassi v1.0* na lateral esquerda com dois perfis de alumínio, porém, no *chassi v2.0* da lateral direita, é possível verificar um único perfil centralizado, tornando assim um visual limpo para o seu chassi, e otimizando espaço para a estrutura da carroceria interna e externa, tornando dessa forma a manutenção tanto mecânica quanto de automação eficiente. No âmbito de utilização consciente de matéria-prima, essa alteração, possibilitou a reutilização do perfil de alumínio retirado da lateral, tornando-o base para o octógono contemplado na Figura 6 [14, 17].

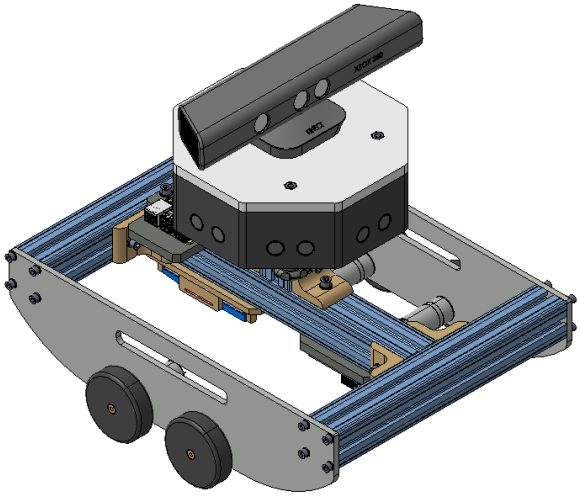

Figure 4: Modelagem 3D DTMR Tupy. Desenho mecânico 3D completo do robô Tupy montado com a integração do módulo de sonares e a câmera Kinect sobre o suporte.

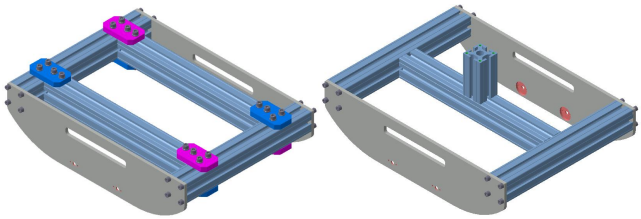

Figure 5: Análise da modelagem 3D do *chassi* v1.0 a esquerda e a análise da modelagem 3D do *chassi* v2.0 a direita. A atualização mostra uma montagem mais simples com o perfil centralizado e ainda utilizando o perfil como base para o octógono de sonares.

Após análise e validação do *chassi* em perfil de alumínio, foi desenvolvido um novo modelo de *design* para o octógono de sonares, com melhorias estruturais e de massa estrutural, facilitando assim, a sua manufatura aditiva para a validação física do componente. Junto com as melhorias físicas estruturais do octógono de sonares, houve também uma melhoria para o sistema de campo de visão para a automação, sendo que são utilizados 8 sonares com ângulo de visão de 15°, tendo uma base com suporte na parte inferior para a fixação nos perfis de alumínio, reduzindo vibrações excessivas nos sonares.

A Figura 6 mostra o octógono de sonares avaliando o posicionamento dos sensores e a parte inferior do octógono com a fixação da base para o perfil de alumínio (tanto base para o encaixe, quanto as furações para passagem de parafusos).

3.1.2 Integração dos periféricos. A integração física do robô Tupy foi realizada principalmente por meio da instalação da plataforma Arduino UNO R3, que utiliza o microcontrolador ATmega328P. Isso se deu devido à sua implementação otimizada com o ROS, utilizando o pacote **rosserial-arduino**. Além disso, a plataforma

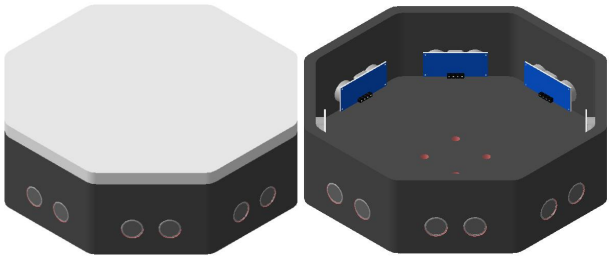

Figure 6: Octógono de Sonares permite o alojamento de até 8 sensores ultrassônicos HC-SR04 e acomodação da plataforma Arduino Uno com a Protoshield, otimizando espaços no robô.

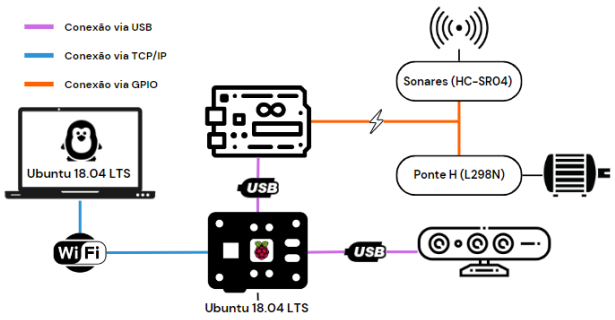

Figure 7: Integração física dos periféricos DTMR Tupy. As conexões entre a plataforma do Arduino, RPi 3B+ e Kinect e são feitas via (USB). A conexão entre o RPi e o *desktop* é feita por (WiFi) no SO Linux Ubuntu 18.04 LTS. Por fim, os sonares e ponte H via (GPIO) do Arduino.

Arduino oferece facilidade na gravação do *firmware*, utilizando o *bootloader* integrado na IDE de código aberto do Arduino.

Adicionalmente, para atender às demandas de comunicação em alto nível e controle remoto, foi incorporado uma plataforma Raspberry Pi 3B+. Isso foi feito para viabilizar a troca de dados por meio do protocolo TCP/IP. A leitura e escrita dos periféricos de baixo nível são realizadas através da comunicação serial (USB) com o ATmega328P. Para controle dos motores e aquisição da leitura dos sonares HC-SR04, a porta GPIO (*General Purpose Input/Output*) é utilizada. A Figura 7 mostra a integração física dos periféricos do DTMR Tupy.

3.2 Sistema Virtual

Após a execução da parte física foi desenvolvida a integração entre os periféricos. Então ocorre a preparação do ambiente para o desenvolvimento da programação, parametrização e virtualização do DTMR do robô Tupy, seguindo a estrutura de desenvolvimento conforme mostra a Figura 8.

Para a seleção dos Sistemas Operacionais nos dispositivos, necessita-se verificar a compatibilidade dos equipamentos para receber o *framework* ROS compatível. Os principais parâmetros a serem considerados, segundo a comunidade do ROS, são as arquiteturas dos

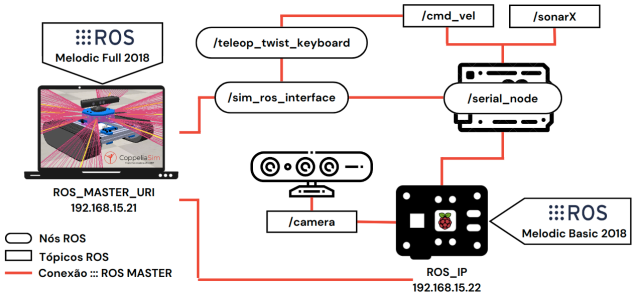

Figure 8: Sistema virtual do DTMR Tupy, as conexões entre os tópicos e os nós pelo ROS Melodic. O *desktop*, como master, troca dados com o RPi e visualiza o Gêmeo Digital Tupy no ambiente virtual do CoppeliaSim.

processadores. Assim foram coletadas as informações dos equipamentos utilizados neste projeto.

- ***Desktop*: arquitetura 64 bits (amd64)**
- **Raspberry Pi 3B+: arquitetura 32 bits (armhf)**

Confirmando a arquitetura do sistema microprocessado, foi definido o *framework* **ROS Melodic Morenia 2018** para realizar a integração entre os diferentes dispositivos. A tabela 1 mostra a compatibilidade das distribuições e arquiteturas.

Table 1: Compatibilidade das Distribuições e Arquiteturas para o ROS Melodic Morenia 2018.

Distro	amd64	arm64	armhf
Linux Ubuntu Artful (17.10)	X		
Linux Ubuntu Bionic (18.04)	X	X	X

Fonte: Adaptado de ROS [20].

Para que o ROS Melodic funcione corretamente, conforme mostra a Tabela 1, a distribuição a ser instalada no Raspberry Pi (armhf) e *Desktop* (amd64) será o **Linux Ubuntu 18.04 LTS Bionic Beaver**.

3.3 Integração

Neste projeto, o microcontrolador ATmega328p tem fundamental importância para o desenvolvimento prático do robô Tupy, pois com ele torna-se possível o controle das rodas através do controle do *driver* da ponte H com o uso de pacotes ROS referentes a tratativa de velocidade linear e angular, e a leitura da distância dos sensores ultrassônicos. Nesta subseção, primeiramente, será exibido a configuração da comunicação da plataforma Arduino com o ROS, e posteriormente, como Publicar e Escrever mensagens.

Para integrar o Arduino ao ROS foi indispensável o uso de determinados pacotes. O principal deles é o *rosserial-arduino*, estabelecendo a comunicação serial RX/TX, tornando-se um "meio" para a troca de mensagens. A partir de então será possível que o ROS reconheça, por exemplo, os tópicos dos sonares (Range) criados via IDE Arduino e que receba as informações da teleoperação *Twist* enviadas pelo terminal *desktop*.

Para realizar as configurações na interface de desenvolvimento do Arduino é imprescindível a inclusão da **Rosserial Arduino Library 0.7.9** no Gerenciador de Bibliotecas Arduino IDE. Com a biblioteca instalada, basta incluir no cabeçalho da programação a função ROS. Também foram utilizadas outras duas bibliotecas, como **NewPing** e **SimpleKalmanFilter**.

3.3.1 Controle Ponte H Motores (Subscriber ROS). O **Twist** é o pacote que envia mensagens geométricas *geometry_msgs* e fornece mensagens geométricas comuns, como: pontos, vetores (lineares e angulares) e posição. Essas mensagens são projetadas para fornecer um tipo de dados comum e facilitar a interoperabilidade em todo o sistema. Com este comando, será possível controlar a direção do Tupy que será teleoperado pelo *desktop*. A instalação deste pacote é simples e deve ser efetuada no *desktop* através do comando **$sudo apt-get install ros-melodic-teleop-twist-keyboard**. A programação do Arduino deve ser feita com base na inscrição do tópico *teleop-twist-keyboard*. Então define-se no cabeçalho da IDE algumas informações pertinente ao tópico.

3.3.2 Sensores Ultrassônicos (Publisher ROS). Assim como as mensagens enviadas pelo pacote *Twist*, temos o pacote *Range*, responsável por enviar mensagens de distância, no caso, o valor da leitura dos sensores ultrassônicos. Foi desenvolvido os nós ROS dos quatro sensores na placa Arduino UNO para visualizar as leituras de distância de cada sensor. Nesta etapa será apresentado, tal qual o controle da Ponte H funcionamento em partes da programação na IDE do Arduino.

Iniciando a programação na IDE Arduino, declaram-se as bibliotecas e funções, algumas definições e variáveis no cabeçalho. Incluem-se as funções **ros.h**, o qual é a biblioteca padrão do ROS, e **ros/time.h**, responsável por sincronizar os tempos ROS. Na sequência, com a função **sensor_msgs/Range.h**, declara o tipo de mensagem ROS que será enviada, no caso distância (*Range*). Por fim, importam-se as bibliotecas **NewPing**, que trabalha com os sensores ultrassônicos HC-SR04, e a **SimpleKalmanFilter**, responsável por filtrar a saída do sensor ultrassônico. Essas bibliotecas devem estar previamente instaladas, conforme visto anteriormente.

Após a declaração das bibliotecas, declaram-se os tipos de variáveis para trabalhar com essas funções indicadas e criam-se objetos **NewPing** e **SimpleKalmanFilter** para todos os sensores, conforme a indicação dos parâmetros comentadas no programa. Para o **NodeHandle** é um objeto que representa que a placa será um nó ROS.

A partir do momento em que o *framework* ROS é iniciado, ele monitora as conexões do nó para enviar as mensagens dos tópicos publicados. Para isso, usa-se o comando **nh.spinOnce( )** para informar ao ROS que uma nova mensagem foi gerada.

3.4 Integração Virtual — Kinect com ROS

A integração do Kinect consiste na instalação dos *drivers* para seu reconhecimento no sistema operacional. Este pacote que será instalado no RPi traz consigo os nós necessários para visualização das imagens coloridas (*RGB color*), imagens de profundidade *depth Image*, IR *Image* e possibilidade do controle do motor do Kinect. Essas poderão ser facilmente visualizados com o *software* RviZ. De modo a agregar ainda mais funções do Kinect neste projeto, após a instalação dos *drivers*, tornará possível a criação do pacote de OpenCV, responsável pelo filtro de imagens.

3.4.1 Instalando drivers *Kinect.* Existem alguns *drivers* disponíveis para o Kinect, porém somente dois deles são dedicados ao uso com o ROS, são eles:

- **OpenNI** https://github.com/ros-drivers/openni_camera
- **Libfreenect** https://github.com/ros-drivers/freenect_stack

No *desktop* foi utilizado o **OpenNI**, pois o **Libfreenect** apresentou alguns erros que não permitiram o êxito na instalação. O uso do **OpenNI** no *desktop* se deu para fins de testes, ou seja, não sendo necessário para a aplicação final do Gêmeo Digital com o robô Tupy. Para o RPi foi instalado o **Libfreenect** sem maiores problemas. Vale salientar que este *drive* é constantemente atualizado diferentemente do OpenNI, que não recebe atualizações há algum tempo.

Como a versão pré-compilada está desatualizada o **libfreenect** foi instalado manualmente a partir do repositório. Dessa forma, tornou-se um processo mais trabalhoso. Por outro lado, foi possível entender o funcionamento da construção dos *drivers* de forma mais clara.

3.4.2 Instalando OpenCV Kinect no RPi. Afim de um melhor aproveitamento do Kinect, buscou se um algorítimo que pudesse traçar as boardas do que estava sendo capturado pelo sensor. Então, buscaram-se referências em Magrin et al. [17] do código para a utilização do **OpenCV**. A instalação do pacote **csf_robotCS_cannyKinect.py** foi realizada no *desktop*. Em seguida, para o ROS reconhecer este arquivo, foi necessário transformá-lo em um pacote ROS.

Após a compilação do pacote, passa-se a configurar o *script* **cannyKinect.py**. Deve-se realizar esse ajuste para que o gerador de imagens passe reconhecer o tipo de tópico a ser lido, publicado pelo *driver Freenect*, e adequar a exibição das imagens, rotacionando e espelhando adequadamente, configurando o arquivo na pasta **catkin_ws/src/img_processor/src**.

3.5 Integração Virtual — ROS com CoppeliaSim

O CoppeliaSim (CS) - EDU, antigo V-REP, permite a simulação virtual deste projeto. O CoppeliaSim traz diversos recursos interessantes nesse sentido, e o principal deles é pelo fato de permitir a interconexão com o ambiente ROS. Como esse estudo baseou-se no artigo de Magrin et al. [17], utilizou-se da modelagem base do robô Tupy, disponível no repositório https://github.com/VRI-UFPR/dtmr.git, para inseri-lo no ambiente virtual.

Algumas modificações foram efetuadas, como parte do aperfeiçoamento da modelagem para simulação. A principal delas, foram as rodas substituídas por esteiras, melhorando a fluidez na simulação.

Para a movimentação controlada por teleoperação, necessitou-se inscrever o tópico **cmd_vel** publicado por **teleop_twist_key**. Na próxima seção será demonstrado como ocorreu a interface de comunicação do ROS com a simulação do robô, configurando os *scripts* **LUA** da programação de integração do robô com o ambiente de simulação.

4 RESULTADOS E DISCUSSÃO

Nesta seção será discutida a proposta *Digital Twin of Mobile Robot* (DTMR), verificando os resultados obtidos pertinentes ao desenvolvimento do projeto, considerando os conceitos da Indústria 4.0, *Cyber-Physical Systems* (CPS), robótica móvel e *Digital Twin* (DT).

4.1 Validação Digital Twin

Esta seção consiste em realizar a validação da aplicação do DTMR no ambiente ciber-físico. Dessa forma, será possível verificar questões pertinentes ao controle da movimentação, as imagens geradas pelo Kinect no *desktop*, os sensores ultrassônicos, bem como, conceber o DT, através da plataforma virtual CoppeliaSim. Finalizado as etapas de integração (*hardware* e *software*), inicia-se o ambiente do *framework* ROS. Primeiramente, é definido que o *desktop* será o ROS MASTER.

4.1.1 Roscore. Comando utilizado para criar uma conexão entre os nós, definindo um ROS MASTER. Sem iniciá-lo, fica impossível estabelecer uma conexão ROS, sendo que só pode ser executado em uma das estações via terminal. Logo, digita-se **roscore** no terminal para iniciar a interface.

4.1.2 Arduino. Após iniciar o **roscore**, deve-se realizar o *login* no robô Tupy via **SSH**. Inicia-se o procedimento para configuração do **rosserial-arduino**. Primeiramente, deve-se habilitar a interface USB, geralmente denominada **ACM***. Essa configuração deve ser realizada, pois, devido à segurança do Linux, deve-se habilitar essa porta para estabelecer a comunicação com dispositivos externos.

Como boas práticas de integração com o ROS, é sempre recomendado que seja exportado em qual dispositivo o ROS MASTER está sendo executado e o IP do dispositivo atual. Esses códigos devem ser utilizados toda vez que alguma aplicação relacionada ao ROS for iniciada. No caso, o MASTER é o *desktop*, e o ROS IP pertence ao robô Tupy.

Em seguida, com o comando **rosrun** carrega o nó **ROS Serial Python NODE**. Dessa forma, o ROS executa o *script* da conexão serial do Arduino na porta **ttyACM0**, publicando os tópicos dos quatro sonares e escrevendo as mensagens de velocidade com o comando **cmd_vel**.

4.1.3 Teleoperação. Para a implementação do controle do robô físico e virtual simultaneamente, utiliza-se o comando **cmd_vel**. Para isto, inicia-se a interface no terminal com comando: **rosrun teleop_twist_keyboard teleop_twist_keyboard.py**

Com as interfaces iniciadas, o pacote então publica as mensagens vetoriais. A partir de então, conforme a captura das teclas, o robô é movimentado com velocidades lineares e angulares definidas.

4.1.4 Kinect e OpenCV. Após a compilação dos *drivers* **freenect**, inicia-se com o **roslaunch** a publicação dos Tópicos da câmera do Kinect, conforme o comando: **roslaunch freenect_launch freenect.launch**

4.2 Visualizando pacotes do ROS com RviZ e rtq_graph

A grande vantagem do uso do RviZ, é a facilidade da visualização de tópicos ativos no ROS. Esta ferramenta é fundamental para a realização de testes durante o desenvolvimento no ambiente ROS, pois se pode verificar que a comunicação entre os nós do ROS está correta e se os *drivers* do Kinect foram iniciados corretamente. A Figura 9 mostra através do detalhe (2) algumas destas opções disponíveis para criação. Já o detalhe (1), é possível visualizar a saída do tópico infravermelho do Kinect.

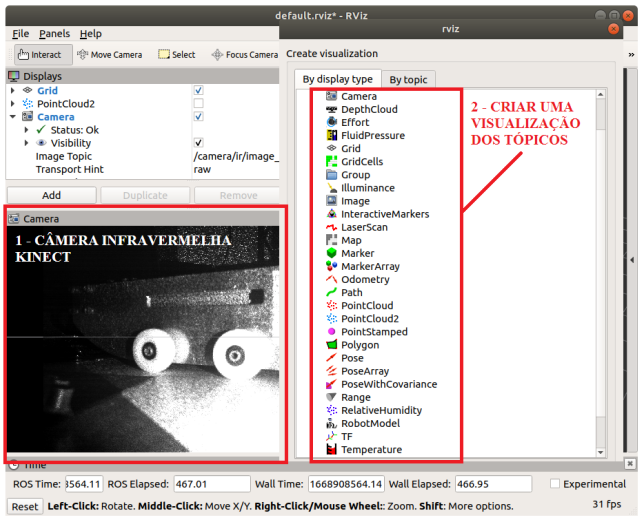

Figure 9: A Figura mostra a saída de imagem (1) no RViz, visualizando o tópico infravermelho da câmera. Dependendo do tipo de sensor utilizado é possível gerar outros tipos de saída (2).

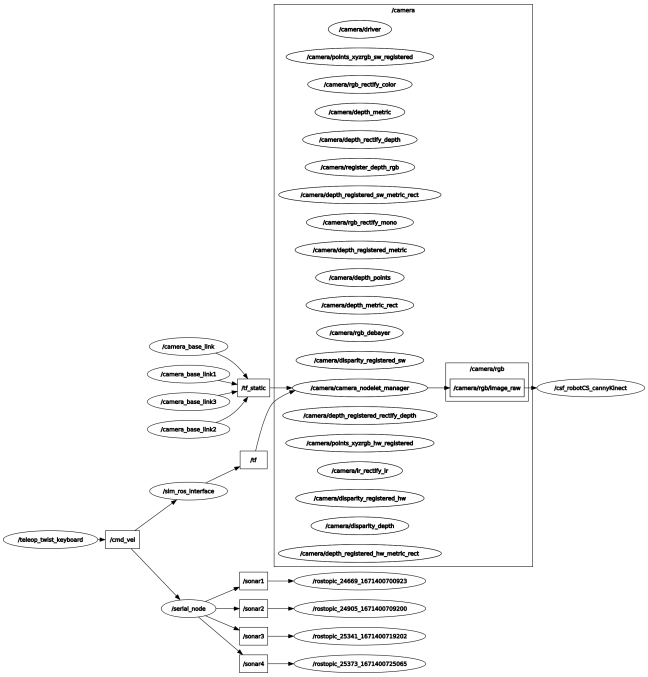

Figure 10: Arquitetura do *framework* ROS, exibindo graficamente os tópicos do robô Tupy com rtq_graph.

Outra forma de visualizar os tópicos ativos, é utilizar a ferramenta gráfica nativa do ROS **rtq_graph**, conforme Figura 10.

A Figura 10 mostra o *driver* Freenect iniciando vários tópicos referentes à câmera. No entanto, apenas o tópico **/camera/rgb** é necessário para o funcionamento do pacote *Canny*. Por outro

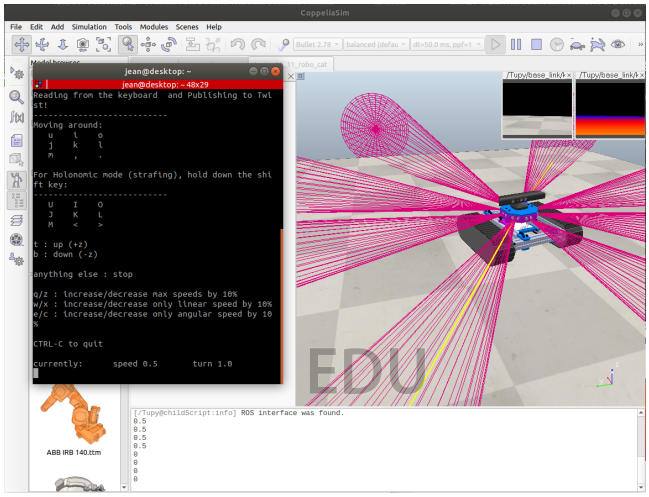

Figure 11: DTMR Tupy no ambiente virtual do CoppeliaSim - EDU no *desktop* e à esquerda, a janela da teleoperação para a movimentação do robô.

lado, o **teleop_twist_keyboard** envia mensagens geométricas através do tópico **cmd_vel** para as interfaces de simulação virtual (**sim_ros_interface**) e o robô Tupy físico através do nó **serial_node** que, também, faz a gestão dos tópicos referentes aos sensores ultrassônicos (sonares), publicando mensagens de distância (*Range*). Dessa forma tem-se o Gêmeo Digital arquitetado pelo *framework* ROS.

4.3 Tupy no ambiente ciber-físico como gêmeo digital

Finalizando todas as etapas demonstradas, procede-se à inicialização do cenário virtual no CoppeliaSim, onde o robô Tupy é modelado e a simulação é iniciada. Com isso, obtém-se o DTMR Tupy no ambiente virtual. Posteriormente, ao inserir comandos no terminal de teleoperação, tanto o robô virtual quanto o físico começam a se movimentar. O pacote de câmera gera imagens do ambiente e, por meio da comunicação serial com o Arduino, as distâncias dos obstáculos são medidas com base nas mensagens de distâncias enviadas pela leitura dos sensores ultrassônicos, conforme mostra a Figura 12 o robô Tupy no ambiente real. Assim, por meio desse desenvolvimento, obtém-se o Gêmeo Digital aplicado à robótica móvel DTMR.

Então, a partir do momento em que comandos são inseridos no terminal da teleoperação, o robô virtual e físico passam a se movimentar, com o pacote da câmera, gera imagens do ambiente e, por fim, através da comunicação serial com o Arduino, medir as distâncias dos obstáculos com as mensagens de distâncias enviadas pela leitura dos sensores ultrassônicos, conforme a Figura 12, do robô Tupy no ambiente real. Portanto, através deste desenvolvimento, tem-se o gêmeo digital, aplicados à robótica móvel DTMR.

Figure 12: DTMR Tupy físico no ambiente real e ao fundo, no monitor, é exibido o pacote *CannyKinect* e abaixo a leitura em "tempo real" de 4 sensores ultrassônicos HC-SR04.

4.4 Melhorias Futuras

Com a coleta de dados pelos testes e resultados aqui apresentados, pode-se colocar diversos pontos para melhorias no projeto, dando continuidade a novas pesquisas e desenvolvimentos com o DTMR.

4.4.1 Locomoção. Propõe-se a adoção de esteiras para a locomoção do robô, com inclusão de odometria, encoders e controle PWM para medição do deslocamento e velocidade dos motores, além de um controle para teleoperação intuitiva.

4.4.2 Percepção. Recomenda-se a inclusão de um sensor LiDAR para mapeamento do ambiente e uma IMU (Unidade de Medição Inercial) para leitura da posição e orientação do robô.

4.4.3 Mapeamento. Propõe-se a implementação de um pacote SLAM (*Simultaneous Localization and Mapping*) para o ROS, permitindo que o robô crie um mapa do ambiente enquanto se desloca, além da modelagem do ambiente real, especificamente o campus UniCuritiba, no ambiente virtual do *software* CoppeliaSim.

4.4.4 Alimentação. Para alimentar o robô, será necessário desenvolver um circuito específico para integrar a bateria com os *powerbanks*, além de uma base para carregamento rápido.

4.4.5 ROS. Para integrar os sistemas físicos e virtuais, recomenda-se a implementação do ROS 2, visando adaptar-se às atualizações fornecidas pelo desenvolvedor e incorporar novos recursos disponíveis.

5 CONCLUSÃO

O resultado do trabalho atendeu a proposta de validar o modelo de um gêmeo digital na área de robótica móvel, DTMR, com o objetivo de aplicar como ferramenta educacional em sistemas ciberfísicos. Utilizando o ROS Melodic, a virtualização desse ambiente foi alcançada com sucesso, abrindo amplas possibilidades de controle e percepção do ambiente. O desenvolvimento da pesquisa reforçou diversos conceitos técnicos e acadêmicos, incluindo o uso de simuladores robóticos, Linux e o ROS, além de ferramentas de desenvolvimento como o Arduino e o Raspberry Pi. Destaca-se também a importância dos robôs móveis em projetos acadêmicos, oferecendo uma plataforma acessível para aplicação de ferramentas de robótica educacional e inclusiva. Com este projeto disponível para continuidade e aprimoramento, busca-se incentivar o compartilhamento de conhecimento dentro da comunidade acadêmica. O presente artigo é concluído afirmando a relevância de elucidar em um experimento prático como é implementado um gêmeo digital com ROS, dada a difícil curva de aprendizado no assunto. Os recursos desenvolvidos no projeto estão disponíveis em: http://github.com/rosie-projects/dtmr_tupy.

REFERENCES

[1] Chimay Anumba, Abiola Akanmu, and John Messner. 2010. Towards a Cyber-Physical Systems Approach to Construction. 528–538. https://doi.org/10.1061/41109(373)53
[2] Radhakisan Sohanlal Baheti and Helen Gill. 2019. Cyber-Physical Systems. *2019 IEEE International Conference on Mechatronics (ICM)* (2019).
[3] Thomas Braunl. 2022. *Embedded robotics : from mobile robots to autonomous vehicles with Raspberry Pi and Arduino* (fourth edition. ed.). Springer, Singapore.
[4] L. Bruzzone and G. Quaglia. 2012. Review article: locomotion systems for ground mobile robots in unstructured environments. *Mechanical Sciences* 3, 2 (2012), 50. https://doi.org/10.5194/ms-3-49-2012
[5] Michelle D. CRUZ. 2010. *Catia V5: Modelagem, Montagem e Detalhamento 2D & 3D* (first ed.). Vol. 1st. Editora Érica.
[6] Luciano Rottava da Silva. 2003. ANÁLISE E PROGRAMAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS DA PLATAFORMA EYEBOT. , 37-38 pages.
[7] G.W. Denardin and C.H. Barriquello. 2019. *Sistemas operacionais de tempo real e sua aplicação em sistemas embarcados.* Blucher. https://books.google.com.br/books?id=FrqxDwAAQBAJ
[8] Rainer Drath and Alexander Horch. 2014. Industrie 4.0: Hit or Hype? [Industry Forum]. *Industrial Electronics Magazine, IEEE* 8 (06 2014), 56–58. https://doi.org/10.1109/MIE.2014.2312079
[9] Lukas Esterle and Radu Grosu. 2016. Cyber-physical systems: challenge of the 21st century. *e and i Elektrotechnik und Informationstechnik* 133 (11 2016), 1–5. https://doi.org/10.1007/s00502-016-0426-6
[10] Cristiane Brasil. FAGALI, Adriano. S.; ULBRICH LIMA. 2009. *Engenharia Integrada por Computador e Sistemas CAD/CAM/CNC - Princípios e Aplicações* (first ed.). Vol. 1st. Editora Artliber.
[11] Kagermann Henning. 2013. Recommendations for implementing the strategic initiative INDUSTRIE 4.0.
[12] Fei HU. 2014. *Cyber-Physical Systems: Integrated Computing and Engineering Design.* 3 pages.
[13] Henning Kagermann, Reiner Anderl, Jürgen Gausemeier, Günther Schuh Wolfgang, Wahlster, and Johannes Winter. 2016. *Industrie 4.0 in a Global Context: Strategies for Cooperating with International Partners (acatech STUDY).*
[14] Jacob. LEAKE, James.; BORGERSON. 2012. *Manual de Desenho Técnico para Engenharia* (second ed.). Editor LTC.
[15] Edward Ashford Lee and Sanjit Arunkumar Seshia. 2016. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach.* 5 pages.
[16] Linux. 2022. What Is Linux. urlhttps://www.linux.com/what-is-linux.
[17] Carlos Eduardo Magrin, Gustavo Del Conte, and Eduardo Todt. 2021. "Creating a Digital Twin as an Open Source Learning Tool for Mobile Robotics. *2021 Latin American Robotics Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR) and 2021 Workshop on Robotics in Education (WRE)* (2021), 13–18. https://doi.org/10.1109/LARS/SBR/WRE54079.2021.9605457
[18] Aaron Martinez and Enrique Fernndez. 2013. *Learning ROS for Robotics Programming.* Packt Publishing.
[19] Elisabete Nakoneczny Moraes. 2013. Método para gerenciamento do consumo de energia elétrica em sistemas ciberfísicos. , 57-59 pages.
[20] Documentation ROS. 2022. ROS/Concepts. urlhttp://wiki.ros.org/ROS/Concepts.
[21] J.B. Sacomano, R.F. Gonçalves, S.H. Bonilla, M.T. da Silva, and W.C. Sátyro. 2018. *Indústria 4.0: conceitos e fundamentos.* Blucher.
[22] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. 2011. *Introduction to Autonomous Mobile Robots* (2nd ed.). The MIT Press.
[23] Ramon Silva Ortigoza, Mariana Marcelino-Aranda, Gilberto Silva Ortigoza, Victor Manuel Hernandez Guzman, Maria Aurora Molina-Vilchis, Griselda Saldana-Gonzalez, Juan Carlos Herrera-Lozada, and Mauricio Olguin-Carbajal. 2012. Wheeled Mobile Robots: A review. *IEEE Latin America Transactions* 6 (2012), 2209–2217. https://doi.org/10.1109/TLA.2012.6418124
[24] S.G. Tzafestas. 2013. *Introduction to Mobile Robot Control.* Elsevier, USA.