



**CENTRO UNIVERSITÁRIO UNICURITIBA
ENGENHARIA DE CONTROLE E AUTOMAÇÃO**

**JEAN FELIPHE DOS SANTOS
JHONI JHOSEP SURDI**

**FERRAMENTA PARA APRENDIZAGEM DE SISTEMAS CIBER FÍSICOS: UMA
APLICAÇÃO INSERIDA NO CONCEITO DE GÊMEO DIGITAL NA ROBÓTICA
MÓVEL**

**CURITIBA
2022/2**

**JEAN FELIPHE DOS SANTOS
JHONI JHOSEP SURDI**

**FERRAMENTA PARA APRENDIZAGEM DE SISTEMAS CIBER FÍSICOS: UMA
APLICAÇÃO INSERIDA NO CONCEITO DE GÊMEO DIGITAL NA ROBÓTICA
MÓVEL**

**Trabalho de Conclusão de Curso apresen-
tado ao Centro Universitário UNICURITIBA
como requisito parcial para obtenção do
Título de Bacharel em Engenharia de Con-
trole e Automação.**

Orientador: Prof. MSc. Carlos Eduardo Magrin

**CURITIBA
2022/2**

Dedicamos este trabalho ao grupo Grupo de Pesquisa em Robótica e Sistemas Embarcados (ROSIE) do curso de Engenharia de Controle e Automação da Unicuritiba, ao professor e orientador MSc. Carlos Eduardo Magrin, ao Dr. Eduardo Todt e ao professor MSc. Jonerlam Roberto Carvalho.

“If you can imagine it, you can do it.” – “Se você pode imaginar, você pode fazer.”

Walter Elias Disney (Walt Disney)

RESUMO

Com o propósito de desenvolver uma ferramenta voltada à aprendizagem de sistemas ciber-físicos, o presente trabalho discorre acerca de robôs móveis que possibilitem a aplicação de conceitos da Indústria 4.0, princípios e pilares norteadores, *Cyber-Physical System* – Sistema Ciber-Físico (CPS) e *Digital Twin* – Gêmeo Digital (DT), com pesquisas bibliográficas relacionadas. Dessa forma, verificou-se que para implementação de um gêmeo digital na robótica móvel, o uso de um Sistema Operacional Robótico era fundamental. A partir disso, foi dada sequência no projeto Tupy, desenvolvido pelo ROSIE Unicuritiba, notando-se a viabilidade de utilização dos recursos pré-disponíveis de integração com as plataformas embarcadas do *Raspberry Pi* (RPI) e do Arduino, através do *framework Robot Operating System* – Sistema Operacional do Robô (ROS). Já para o processo de virtualização do robô, necessita-se o uso de tecnologias de computação gráfica. O conjunto de estudos, resultou no desenvolvimento de um *Digital Twin Mobile Robot* – Gêmeo Digital Robô Móvel (DTMR) do robô Tupy. Apresentando assim resultados relevantes como: a troca de informações de dois diferentes tipos de plataformas de desenvolvimento (Arduino e RPI), a integração do sistema operacional ROS no robô móvel Tupy, a teleoperação e visualização efetiva do robô móvel Tupy no simulador de robótica CoppeliaSim. Dessa forma, conclui-se que esse trabalho é uma aplicação funcional da pesquisa de virtualização proposta por Magrin, Del Conte e Todt (2021) de um DTMR.

Palavras-chave: Sistemas Ciber-Físicos, Gêmeo Digital, ROS, Simulação e Robótica Móvel.

ABSTRACT

With the purpose of developing a tool aimed at learning cyber-physical systems, the present work discusses mobile robots that application of Industry 4.0 concepts, principles, and pillars cyber-physical system (CPS) and Digital Twin (DT), with related bibliographic research. Thus, it was found that, for the implementation of a digital twin in mobile robotics, the use of a Robotic Operational theme was key. From this, a sequence was given in the Tupy project, developed by the Research Group in And Embedded Systems (ROSIE) Unicuritiba, noting the feasibility of using the resources pre-available integration with Raspberry Pi (RPI) and Arduino embedded platforms, through the Robot Operating System framework -Robot operating system (ROS). For the robot virtualization process, the use of computer graphics technologies is required. The set of studies resulted in the development of a Tupy robot DTMR. Thus presenting relevant results such as the exchange of information from two different types of development platforms (Arduino and RPi, the integration of the operating system ROS into the Tupy mobile robot, the teleoperation and effective visualization of the Tupy mobile robot in the CoppeliaSim robotics simulator. In this way, it is concluded that this work is a functional application of the virtualization research proposed by Magrin, Del Conte e Todt (2021).

Keywords: Cyber-Physical Systems, Digital Twin, ROS, Simulation and Mobile Robotics.

LISTA DE FIGURAS

Figura 1 — Etapas das Revoluções Industriais.	28
Figura 2 — Os seis princípios da Indústria 4.0.	29
Figura 3 — Sistemas ciber-físicos.	31
Figura 4 — Áreas da Robótica.	32
Figura 5 — Fluxo <i>loop sensorimotor</i>	34
Figura 6 — Classificação de Robôs em relação às máquinas.	35
Figura 7 — Conceito de Robótica Educacional.	36
Figura 8 — Relações entre Robótica educacional e Educação inclusiva. .	38
Figura 9 — Kits Robóticos educacionais.	39
Figura 10— Robô Móveis educacionais: ROME (Esquerdo) e OCTO (Direito).	39
Figura 11— Disciplinas que constituem a mecânica: estática, cinemática e dinâmica.	40
Figura 12— Relações dos sistemas mecânicos.	41
Figura 13— Mecanismos de locomoção utilizados em sistemas biológicos.	42
Figura 14— Categorias de Robôs móveis terrestres.	44
Figura 15— Exemplo de Robô Híbrido com sistema LW.	44
Figura 16— Exemplos de pernas mecânicas com três graus de liberdade.	45
Figura 17— Nanokhod — Exemplo de robô com esteiras.	46
Figura 18— Plano de referência global e plano de referência local do robô.	48
Figura 19— Cinemática de robôs e seus elementos de interesse.	49
Figura 20— Cinemática do Robô Tupy.	50
Figura 21— Exemplo estrutural de um Termoplástico semicristalino.	51
Figura 22— Composição Química do Ácido Polilático PLA.	52
Figura 23— Etapas de um processo genérico de Manufatura Aditiva.	53
Figura 24— Exemplo de um sistema para fabricação por FDM/FFF.	55
Figura 25— Robô TUPY-4WD versão 1, vista completa e vista em explosão robô TUPY-4WD.	56
Figura 26— Elementos típicos de um sistema embarcado.	57
Figura 27— Arduino UNO R3 e Raspberry Pi 3b+.	58
Figura 28— Representação das camadas do modelo OSI e o método de comunicação	59
Figura 29— Diferenças entre modelo OSI e TCP/IP	61
Figura 30— <i>Hardware</i> Kinect.	64
Figura 31— Diagramas Ponte H com <i>software</i> Proteus.	65

Figura 32—Conceito Arquitetura ROS	68
Figura 33—Arquitetura ROS nível gráfico computacional.	69
Figura 34—Exemplo de rede ROS ativa com <i>rqt_graph</i>	70
Figura 35—Utilização das Ferramentas CAD/CAE/CAM.	71
Figura 36— Robô TUPY-4WD versão 1, vista completa e vista em explosão robô TUPY-4WD.	72
Figura 37—Exemplo de Simulação no CoppeliaSim.	74
Figura 38— <i>Scripts</i> nativos do CoppeliaSim.	75
Figura 39—Estrutura de controle V-REP/CoppeliaSim.	76
Figura 40—Estrutura do Filtro de Kalman.	81
Figura 41—Esquema do processamento de imagem computacional.	81
Figura 42—Exemplo do algoritmo de Canny aplicado.	82
Figura 43—Arquitetura CPS com DT.	83
Figura 44—Proposta de um gêmeo digital utilizando o ROS	84
Figura 45—Tríade DTMR Tupy	85
Figura 46—Fluxograma geral de integração DTMR Tupy.	85
Figura 47—robô Tupy versão 1	86
Figura 48 —Estrutura do desenvolvimento para o sistema mecânico do DTMR Tupy	87
Figura 49—Seleção de tipo de arquivo STEP para o SolidWorks 2021. ..	88
Figura 50— Seleção de propriedades de importação STEP no SolidWorks 2021.	88
Figura 51— Estrutura para a construção do modelo 3D no SolidWorks do DTMR Tupy.	89
Figura 52—CAD — Suporte para sonares.	90
Figura 53—CAD 3D — suporte e base para octógono.	90
Figura 54— CAD 3D — Suporte RPi (esquerda) / Tampa para octógono (di- reita).	91
Figura 55—CAD 3D — montagem final do robô Tupy.	91
Figura 56—Ferramenta de exportação SolidWorks para URDF.	92
Figura 57 —Configuração de juntas e ligações do robô no exportador URDF.	93
Figura 58—Modelo virtual DMTR Tupy no CoppeliaSim.	93
Figura 59—Estrutura do sistema eletrônico do DTMR Tupy.	94
Figura 60—Arduino e Protoshiled.	95
Figura 61— Fluxograma de integração: Arduino — Ponte H L298N — Moto- res.	95

Figura 62—Fluxograma de integração Arduino — Sonares (HC-SR04).	96
Figura 63— Fluxograma de integração: <i>Desktop</i> , Arduino e RPi 3B+ via USB.	97
Figura 64 —Fluxograma de integração: <i>desktop</i> , Arduino e RPi 3B+ via GPIO.	98
Figura 65—Fluxograma de integração: Kinect — RPi.	98
Figura 66— Impressora 3D Creatily Ender 3.	99
Figura 67— Peças finalizadas.	100
Figura 68—Acabamento da furação dos sensores após impressão.	100
Figura 69—Furação ajustada.	101
Figura 70—Montagem dos sensores HC-SR04 e placa Arduino UNO	102
Figura 71—Montagem Kinect na tampa.	102
Figura 72—Fixação do suporte octogonal no chassi do robô Tupy.	103
Figura 73—Ligação dos sonares no suporte octogonal	104
Figura 74—Ligação elétrica placa RPi	104
Figura 75—Integração: Arduino — Sonares (HC-SR04).	105
Figura 76—Fonte.	106
Figura 77—Integração: Arduino — Sonares (HC-SR04).	106
Figura 78—Estrutura sistema virtual do DTMR Tupy.	107
Figura 79—Gravação na plataforma Balena Etcher.	109
Figura 80—Cartão Micro SD inserido no RPi.	109
Figura 81—Conexão SSH do <i>desktop</i> para o robô Tupy.	111
Figura 82—Estrutura de Arquivos do <i>catkin workspace</i> .	112
Figura 83—Estrutura de Arquivos do <i>catkin workspace</i> do Tupy.	112
Figura 84—Pacotes Rosserial Instalados.	113
Figura 85—Gerenciador de Biblioteca Arduino IDE.	114
Figura 86—Biblioteca Utilizadas na Programação do Arduino.	114
Figura 87—Definição iniciais da Ponte H na IDE Arduino.	115
Figura 88—Programação básica para teleoperação com ponte H.	116
Figura 89—Configuração final da ponte H na IDE Arduino.	117
Figura 90—Cabeçalho Sensores Ultrassônicos na IDE Arduino.	118
Figura 91—Função <code>sensorCycle ()</code> .	119
Figura 92—Função <code>echoCheck</code> .	119
Figura 93—Função <code>oneSensor</code> .	119
Figura 94—Função <code>returnLastValidRead</code> .	120
Figura 95—Função <code>applyKF</code> .	120
Figura 96—Funções de Tempo.	120

Figura 97—Escrevendo mensagens para o ROS.	121
Figura 98—setup().	122
Figura 99—loop().	122
Figura 100—Códigos para instalação dos drivers do Kinect.	124
Figura 101—Códigos Git-Clone para drivers do Kinect.	124
Figura 102—Códigos para criar o pacote Canny Kinect.	125
Figura 103—Configuração do pacote Canny Kinect.	126
Figura 104—Árvore projeto CoppeliaSim.	127
Figura 105—sysCall_init.	127
Figura 106—sysCall_actuation.	128
Figura 107—sysCall_cleanup.	128
Figura 108—Iniciando Roscore no Terminal <i>desktop</i>	130
Figura 109—Configurando Rosserial-Arduino no Terminal.	131
Figura 110—Iniciando Rosserial-Arduino no Terminal.	132
Figura 111—Rostopic List.	132
Figura 112—Echo 4 Sonares.	133
Figura 113—Iniciando Teleoperação.	134
Figura 114—Iniciando Pacote Freenect.	135
Figura 115—Canny Kinect em execução.	136
Figura 116—Visualizando o Tópico infravermelho da câmera.	137
Figura 117—Arquitetura dos tópicos do robô Tupy com rtq_graph.	138
Figura 118—Tupy no ambiente CoppeliaSim.	141
Figura 119—Tupy no ambiente Real.	142
Figura 120—Estrutura DTMR Tupy	143
Figura 121—Resultado da montagem mecânica.	144
Figura 122—Montagem mecânica com as peças desenvolvidas.	145
Figura 123—Montagem mecânica dos componentes eletrônicos.	145
Figura 124—Pontos para melhoria	146
Figura 125—Problemas Placa RPi	147
Figura 126—Resultado da modelagem do robô Tupy.	149

LISTA DE TABELAS

Tabela 1 —Pontos-chave para Locomoção.	43
Tabela 2 —Exemplos de sistemas com rodas.	47
Tabela 3 —Comparativo entre exemplos de comunicação serial.	62
Tabela 4 —Exemplos de sinais de sensores utilizados na robótica.	63
Tabela 5 —Exemplo de composição de um sistema computacional.	66
Tabela 6 — Compatibilidade das Distribuições e Arquiteturas para o ROS Melodic Morenia 2018.	108

LISTA DE ABREVIATURAS E SIGLAS

AI *Artificial Intelligence* – Inteligência Artificial

AM *Additive Manufacturing* – Manufatura Aditiva

AMR *Autonomous Mobile Robots* – Robôs Móveis Autônomos

ANSI *American National Standards Institute* – Instituto Nacional de Padrões Americano

API *Application Programming Interface* – Interface de Programação de Aplicativos

AR *Argumented Reality* – Realidade Argumentada

ARES *Advanced Routing and Editing software* – Software Avançado de Roteamento e Edição

BCI *Brain-computer Interface* – Interface Cérebro-Computador

CAD *Computer-Aided Design* – Desenho Auxiliado por Computador

CAE *Computer-Aided Engineering* – Engenharia Auxiliada por Computador

CAM *Computer-Aided Manufacturing* – Manufatura Auxiliada por Computador

CATIA *Computer-Aided Three-Dimensional Interactive Application* – Aplicativo Interativo Tridimensional Auxiliado por Computador

CC *Clouding Computer* – Computação em Nuvem

CFD *Computational Fluid Dynamics* – Dinâmica de Fluidos Computacional

CNC *Computer Numerical Control* – Controle Numérico Computadorizado

CPS *Cyber-Physical System* – Sistema Ciber-Físico

DNS *Domain Name System* – Sistema de Nome de Domínio

DT *Digital Twin* – Gêmeo Digital

DTMR *Digital Twin Mobile Robot* – Gêmeo Digital Robô Móvel

ER *Educational Robotics* – Robótica Educacional

FDM *Fused deposition modeling* – Modelagem de Deposição Fundida

FEA *Finite Element Analysis* – Análise de Elementos Finitos

FFF *Field-flow fractionation* – Fracionamento de Fluxo de Campo

FTP *File Transfer Protocol* – Protocolo de Transferência de Arquivos

GND *Ground* – Terra

GPIO *General Purpose Input/Output* – Entrada/Saída de Uso Geral

GUI *Graphical User Interface* – Interface Gráfica do Usuário

H2M *Human to Machine* – Homem para Máquina

HCI *Human Computer Interaction* – Interação Homem-Computador

HRI *Human Interaction Robots* – Robôs de Interação Humana

HTML *HyperText Markup Language* – Linguagem de Marcação de Hipertexto

HTTP *Hypertext Transfer Protocol* – Protocolo de Transferência de Hipertexto

IA *Inteligência Artificial*

IDE *Integrated Development Environment* – Ambiente de Desenvolvimento Integrado

IEEE *Institute of Electrical and Electronics Engineers* – Instituto de Engenheiros Eletricistas e Eletrônicos

IoS *Internet of Services* – Internet de Serviços

IMU *Inertial Measurement Units* – Unidade de Medição Inercial

IoT *Internet of Things* – Internet das Coisas

IP *Internet Protocol* – Protocolo de Internet

IPv4 *Internet Protocol Version 4* – Protocolo de Internet Versão 4

IR *Infrared* – Infravermelho

ISIS *Intelligent Schematic Input System* – Sistema de Entrada Esquemática Inteligente

ISO *International Organization for Standardization* – Organização Internacional para Padronização

L *Legged Locomotion* – Locomoção por Pernas

LET *Lean Education Technology* – Tecnologia de Educação Enxuta

LiDAR *Light Detection and Ranging* – Detecção por Luz e Distanciamento

LM *Layered Manufacturing* – Fabricação em Camadas

LT *Leg-Track* – Perna-Esteira

LW *Leg-Wheel* – Perna-Roda

LWT *Leg-Wheel-Track* – Perna-Roda-Esteira

M2M *Machine to Machine* – Máquina para Máquina

MAC *Media Access Control* – Controle de Acesso de Mídia

NC *Numeric Control* – Controle Numérico

NFS *Network File System* – Sistema de Arquivos de Rede

OOP *Object-oriented programming* – Programação Orientada a Objetos

OSI *Open Systems Interconnection* – Interconexão de Sistemas Abertos

OSS *Open-Source Software* – Software de Código Aberto

PA *Polyamide* – Poliamida

PBL *Project-Based Learning* – Aprendizagem Baseada em Projetos

PC *Polycarbonate* – Policarbonato

PDM *Product data management* – Gerenciamento de Dados do Produto

PE *Polietilo* – Polietileno

PES *Polyesters* – Poliéster

PIC *Peripheral Interface Controller* – Controlador de Interface Periférica

PLA *Polylactic acid* – Ácido Polilático

PMMA *Poly Methyl Methacrylate* – Polimetilmetacrilato

PP *Polypropylene* – Polipropileno

PPO *Polyphenol Oxidase* – Polifenol Oxidase

PS *Polystyrene* – Poliestireno

PTFE *Politetrafluoretileno* – Politetrafluoretileno

PVC *Polyvinyl* – Polivinil

PWM *Pulse Width Modulation* – Modulação por Largura de Pulso

R in E *Robotics in Education* – Robótica na Educação

RFID *Radio Frequency Identification* – Identificação por Rádio Frequência

RGB *Red Green Blue* – Vermelho, Verde e Azul

ROS *Robot Operating System* – Sistema Operacional do Robô

ROSIE Grupo de Pesquisa em Robótica e Sistemas Embarcados

RP *Rapid Prototyping* – Prototipagem Rápida

RPi *Raspberry Pi*

RTOS *Real Time Operating System* – Sistema Operacional em Tempo Real

Rviz *ROS Visualization* – Visualização ROS

RX *Receive* – Receber

SD *Secure Digital* – Secure Digital

SFF *Solid Freeform Fabrication* – Fabricação Sólida de Forma Livre

SLAM *Simultaneous Localization and Mapping* – Localização e Mapeamento Simultâneos

SMTP *Simple Mail Transfer Protocol* – Protocolo Simples de Transferência de Correio

SNMP *Simple Network Management Protocol* – Protocolo Simples de Gerenciamento de Redes

SO *Sistema Operational* – Sistema Operacional

SSH *Secure Shell* – Protocolo de Rede Criptográfico

STEAM *Science, Technology, Engineering, Arts, and Mathematics* – Ciência, Tecnologia, Engenharia, Artes e Matemática

STEP *Standard for the Exchange of Product Data* – Norma para Troca de Dados de Produtos

STL *Standard Template Library* – Biblioteca de Modelos Padrão

T *Tracked Locomotion* – Locomoção por Esteiras

TCP/IP *Transmission Control Protocol/Internet Protocol* – Protocolo de Controle de Transmissão/Protocolo de Internet

TI Tecnologia da Informação

TTL *Transistor-Transistor logic* – Lógica Transistor-Transistor

TX *Transmit* – Transmitir

URDF *Unified Robot Description Format* – Formato de Descrição do Robô Unificado

USB *Universal Serial Bus* – Barramento Serial Universal

VR *Virtual reality* – Realidade Virtual

V-REP *Virtual Robot Experimentation Platform* – Plataforma de Experimentação de Robôs Virtuais

VSM *Virtual System Modelling* – Modelagem de Sistema Virtual

W *Wheeled Locomotion* – Locomoção por Rodas

WLAN *Wireless Local Area Network* – Rede Local Sem Fio

WPAN *Wireless Personal Area Network* – Rede de Área Pessoal Sem Fio

WT *Wheel-Track* – Esteira-Roda

XML *Extensible Markup Language* – Linguagem de Marcação Estendida

SUMÁRIO

1 INTRODUÇÃO	24
1.1 PROBLEMA	25
1.2 HIPÓTESE	25
1.3 JUSTIFICATIVA	25
1.4 OBJETIVO GERAL	26
1.5 OBJETIVOS ESPECÍFICOS	26
1.6 METODOLOGIA	26
2 REFERENCIAL TEÓRICO	28
2.1 INDÚSTRIA 4.0	28
2.2 SISTEMAS CIBER-FÍSICOS	31
2.3 ROBÓTICA MÓVEL	32
2.3.1 Sistema Mecânico	40
2.3.1.1 Sistemas de Locomoção	42
2.3.1.2 Locomoção para Robôs móveis terrestres	43
2.3.1.3 Cinemática do Robô Móvel	47
2.3.1.4 Materiais e Métodos de fabricação	50
2.3.2 Sistema Eletrônico	56
2.3.2.1 Sistemas Embarcados	56
2.3.2.2 Comunicação	58
2.3.2.3 Percepção	63
2.3.2.4 Periféricos de saída: módulos, acionamentos e alimentação	65
2.3.3 Sistemas Computacionais	66
2.3.3.1 Sistemas Operacionais	67
2.3.3.2 Modelagem Computacional 3D	71
2.3.3.3 Linguagens de Programação.	77
2.3.3.4 Filtros Digitais	80
2.4 GÊMEO DIGITAL	82
3 DESENVOLVIMENTO	85
3.1 SISTEMA FÍSICO	86
3.1.1 Implementação de melhorias na plataforma pré-desenvolvida	86
3.1.2 Integração Física	93
3.1.2.1 Integração: Arduino e Protoshield V5	94
3.1.2.2 Integração: Arduino e Ponte H (L298N) com Motores Elétricos	95
3.1.2.3 Integração: Arduino com Sonares (HC-SR04)	96
3.1.2.4 Integração: Arduino com RPi 3B+	97

3.1.2.5	Integração: Kinect com RPi 3B+	98
3.1.3	Montagem eletromecânica do robô Tupy	99
3.1.3.1	Mecânica	99
3.1.3.2	Eletrônica	103
3.2	SISTEMA VIRTUAL	107
3.2.1	Seleção do Sistema Operacional nos dispositivos	107
3.2.2	Instalação do SO no <i>desktop</i>: Arquitetura 64 bits (amd64)	108
3.2.3	Instalação do SO no RPi 3B+: Arquitetura 32 bits (armhf)	108
3.2.4	Interface de comunicação <i>desktop</i> com RPi	109
3.2.4.1	Endereçamento IP dos dispositivos	110
3.2.4.2	Acesso remoto	110
3.2.5	Instalação do Framework ROS Melodic	111
3.2.5.1	<i>desktop</i> : ros-melodic-desktop-full	111
3.2.5.2	RPi 3B+: ros-melodic-base	111
3.2.6	Criação de <i>workspace catkin</i>	112
3.2.7	Integração Virtual — Arduino com ROS	112
3.2.7.1	Comunicação Arduino com ROS	113
3.2.7.2	Controle Ponte H Motores (<i>Subscriber</i> ROS)	115
3.2.7.3	Sensores Ultrassônicos (<i>Publisher</i> ROS)	117
3.2.7.4	Carregando o <i>sketch</i> no microcontrolador	123
3.2.8	Integração Virtual — Kinect com ROS	123
3.2.8.1	Instalando drivers Kinect	123
3.2.8.2	Instalando OpenCV Kinect no RPi	124
3.2.9	Integração Virtual — ROS com CoppeliaSim	126
4	VALIDAÇÃO	129
4.1	VALIDAÇÃO DO DTMR TUPY	129
4.1.1	Ambiente ROS	129
4.1.1.1	Roscore	129
4.1.1.2	Arduino	130
4.1.1.3	Teleoperação	133
4.1.1.4	Kinect e OpenCV	134
4.1.2	Visualizando pacotes do ROS com Rviz e rtq_graph	136
4.1.3	Resumo da inicialização dos serviços do Gêmeo Digital	139
4.1.4	Tupy no ambiente ciber-físico como gêmeo digital	140
5	RESULTADOS E DISCUSSÃO	143
5.1	DISCUSSÃO SOBRE OS RESULTADOS DO SISTEMA FÍSICO	143

5.1.1 Mecânica	143
5.1.2 Eletrônica	147
5.1.3 Computação	148
5.2 DISCUSSÃO SOBRE OS RESULTADOS DO SISTEMA VIRTUAL	149
5.2.1 Mecânica	149
5.2.2 Eletrônica	150
5.2.3 Computação	151
6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS	153
6.1 CONSIDERAÇÕES FINAIS E CONCLUSÃO.	153
6.2 PROPOSTAS PARA TRABALHOS FUTUROS	153
REFERÊNCIAS	155
APÊNDICE A – ESQUEMÁTICO PROTEUS INTEGRAÇÃO FÍSICA	163
APÊNDICE B – PROGRAMA ARDUINO PONTE H COM ROS	165
APÊNDICE C – PROGRAMA ARDUINO 4 SENSORES ULTRASSÔNICOS COM ROS	167
APÊNDICE D – PROGRAMA ARDUINO COMPLETO ROBÔ TUPY	169
APÊNDICE E – PROGRAMA EM PYTHON 2.7 CANNY KINECT	171

1 INTRODUÇÃO

A robótica já percorreu um longo caminho. Entretanto, para os robôs móveis, em particular, está ocorrendo uma tendência semelhante à observada para os sistemas de computador: a transição da computação de mesa para dispositivos portáteis. No passado, os robôs móveis eram controlados por grandes sistemas de computador que não podiam ser transportados facilmente. Hoje, podemos construir pequenos robôs móveis com inúmeros atuadores e sensores controlados por dispositivos de baixo custo, pequenos e com sistemas de computador embarcados (BRAUNL, 2022).

Um projeto otimizado de plataforma robótica móvel pode atender a vários estágios de conhecimento, pois demandam de habilidades relacionadas à mecânica, eletrônica e programação. A criação de um ambiente virtual para projeto de robôs envolve o uso de soluções mecânicas *Computer-Aided Design* – Desenho Auxiliado por Computador (CAD), que permitem a aceleração dos testes do protótipo antes mesmo de iniciar a montagem do robô físico. O conceito de gêmeo digital transforma o desenvolvimento da robótica móvel em uma ferramenta de aprendizagem que atende a diferentes níveis de conhecimento e apoia trazer colaboradores de outras áreas juntos (MAGRIN; DEL CONTE; TODT, 2021).

Uma cópia virtual DT do mundo físico, que permite testar e simular eventos ou processos com maior segurança e menor custo do que se realizados no mundo físico. Todas as aplicações que utilizam as arquiteturas CPS são formadas por duas camadas: camadas de tecnologia operacional (física); e camada virtual, de aplicações de tecnologia da informação (ciber). No entanto, os protocolos de comunicação nessas arquiteturas podem diferir de modelos de informações usados na tecnologia da informação, sendo mais próximas dos protocolos de automação (SACOMANO, 2018).

No contexto acadêmico, como instrumento de aprendizagem, o uso da robótica móvel com ênfase em sistemas ciber-físicos e Indústria 4.0, demonstra-se como um meio para os alunos compreenderem o funcionamento efetivo destes sistemas. Assim, a criação de um DTMR proporciona esse desenvolvimento, pois consiste no uso de diversas ferramentas, como software voltado de modelagem CAD/*Computer-Aided Engineering* – Engenharia Auxiliada por Computador (CAE)/*Computer-Aided Manufacturing* – Manufatura Auxiliada por Computador (CAM) (SolidWorks), simuladores robóticos (CoppeliaSim), plataformas embarcadas (Raspberry Pi e Arduino), sistemas operacionais computacionais (Linux) e robóticos (ROS). Através destes,

projeta-se ambientes virtuais para validar, a operacionalidade de um robô, testando locomoção, percepção e tomadas de decisão clonando um sistema físico.

1.1 PROBLEMA

Como um robô móvel consegue ter seu sistema físico teleoperado por seu DT?

1.2 HIPÓTESE

Se desenvolvermos o DT do robô Tupy, teremos um robô móvel teleoperado capaz de prover aplicações como: a de transmissão de dados em tempo real de um sistema físico com sua contraparte virtual, a de um robô sendo controlado remotamente por um simulador robótico, e de fomento para criação de futuras aplicações voltadas tanto para a área acadêmica quanto para a industrial.

1.3 JUSTIFICATIVA

Ao observar os crescentes avanços tecnológicos, impulsionados pela indústria 4.0 nos últimos anos, bem como a demanda progressiva de produtos e pesquisas baseadas em DTs inseridos nos CPS ao nível global, nota-se a viabilidade de elaborar um projeto de pesquisa com ênfase na criação de uma ferramenta versátil e de baixo custo composto por um robô móvel.

Além disso, há outros pontos que compõem a motivação para este projeto, como a análise para a engenharia do controle e automação em relação ao desenvolvimento de robôs através do simulador robótico CoppeliaSim, e da aplicação do *frameworks* ROS para o desenvolvimento de softwares voltados para a robótica.

Este projeto é um precedente da pesquisa acadêmica proposto por Magrin, Del Conte e Todt (2021), tendo como objetivo principal a virtualização do robô Tupy. Assim, para a implementação de novas peças mecânicas, faz-se o uso de conceitos da engenharia mecânica, estudos de polímeros da engenharia, e processos de manufatura aditiva. Para a reestruturação do sistema de controle e sensoriamento do robô, utilizam-se estudos da engenharia elétrica e eletrônica, robótica, protocolos de comunicação, e de eletrônica embarcada. Para a integração dos sistemas virtuais, o uso de conceitos de sistemas operacionais, *frameworks*, e linguagens de

programação. Pela virtualização, o emprego de conceitos de simuladores e integradores.

Dessa maneira, este projeto colabora com a comunidade científica por apresentar na prática o conceito de gêmeo digital, da indústria 4.0, dos sistemas ciberfísicos, da manufatura aditiva, da simulação, e de desenvolvimento através de um sistema operacional dedicado a robótica.

1.4 OBJETIVO GERAL

O objetivo geral deste trabalho é propor uma aplicação prática do artigo “*Creating a Digital Twin as an Open Source Learning Tool for Mobile Robotics*”, publicado no *Institute of Electrical and Electronics Engineers* – Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) Xplore por Magrin, Del Conte e Todt (2021).

1.5 OBJETIVOS ESPECÍFICOS

Os objetivos específicos necessários para solucionar o problema proposto e atingir o objetivo geral são:

- Pesquisar sobre conceitos fundamentais para a virtualização do robô Tupy, sendo Indústria 4.0, CPS, robótica móvel e DT;
- Projetar a implementação CPS no robô Tupy original, alicerçados sobre conceitos DT;
- Desenvolver algoritmo para controle em linguagem de programação (C e Python) e integração entre plataforma Arduino, Raspberry e *desktop*, através do *framework* ROS;
- Validar controle remoto da movimentação simultânea e a percepção do ambiente, com o processamento computacional em tempo real do robô Tupy;

1.6 METODOLOGIA

Este trabalho utiliza-se de pesquisa aplicada e de natureza exploratória em relação ao desenvolvimento de gêmeos digitais pelo uso de robôs móveis aplicados aos conceitos dos Sistemas Ciber-Físicos. Inicia-se com uma pesquisa bibliográfica sobre a composição da indústria 4.0, e dos campos que envolvem a robótica

móvel. Em seguida apresenta-se o desenvolvimento baseado no método hipotético-dedutivo, composto por testes e validações do sistema físico (modelagem do robô, manufatura de peças mecânicas, montagem e integração de componentes eletrônicos) e do sistema virtual (integração de interfaces operacionais, integração de *framework ROS*, e integração com o simulador CoppeliaSim). Finalizando com uma abordagem quantitativa, que agrupa dados em relação à montagem final e referentes ao sistema virtual e físico que compõem o gêmeo digital.

2 REFERENCIAL TEÓRICO

Para realizar o desenvolvimento de um robô móvel inserido no conceito de gêmeo digital se faz necessária uma pesquisa ampla acerca de conhecimentos que constituem duas grandes áreas de estudo: a robótica móvel e a indústria 4.0. Para a robótica móvel conceitos de mecânica, elétrica, eletrônica e itens relacionados a computação. Em relação à indústria 4.0, conceitos dos CPS e de DT.

2.1 INDÚSTRIA 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, consideradas assim como o início de uma quarta revolução industrial (DRATH; HORCH, 2014) e (KAGERMANN, 2016). Segundo (HENNING, 2013), 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 Figura 1, mostra como se deu cada revolução, a partir de contextos específicos de cada época. As três primeiras revoluções, se desenvolveram em um ambiente físico, ou seja, com a centralização dos processos da manufatura. Já a atual, quarta revolução, traz todo esse contexto para o ambiente digital e descentraliza os serviços, através dos CPS, *Internet of Things* – Internet das Coisas (IoT) e *Internet of Services* – Internet de Serviços (IoS) (SACOMANO, 2018).

Figura 1: Etapas das Revoluções Industriais.

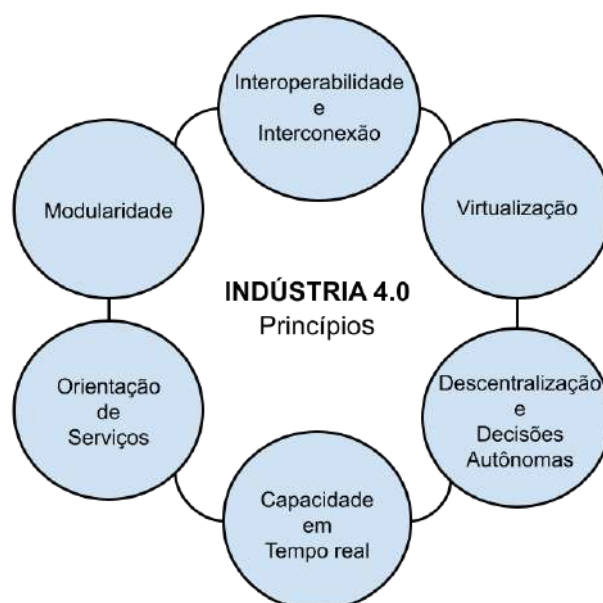


Fonte: Adaptado de Indústria 4.0: conceitos e fundamentos, Sacomano (2018).

São diversas as fontes que abordam sobre a indústria 4.0, mas poucas que explicam com clareza quais seus pilares, assim utiliza-se os mais citados e utilizados na comunidade acadêmica (FALCÃO, 2019): os pilares da indústria 4.0, segundo Hermann, Pentek e Otto (2016) e os nove pilares tecnológicos da indústria 4.0 segundo Rüßmann (2016).

De acordo com (HERMANN; PENTEK; OTTO, 2016) a Figura 2 mostra os seis princípios da indústria 4.0: interoperabilidade e interconexão, capacidade em tempo real, virtualização, descentralização e decisões autônomas e orientação a serviços.

Figura 2: Os seis princípios da Indústria 4.0.



Fonte: Adaptado de *Smart factories — The product of Industry 4.0* Ilanković (2020).

- Interoperabilidade e Interconexão: É definida como a capacidade da troca e uso de informações entre dois ou mais sistemas, ou componentes (GERACI, 1991). No contexto da indústria 4.0 a interoperabilidade envolverá o acesso aos dados em tempo real e a sincronia entre as máquinas inteligentes, as instalações de armazenamento e as interações Humano-Máquina (*Human to Machine — Human to Machine – Homem para Máquina (H2M)*), esta cadeia que forma os CPS (CARVALHO; CAZARINI, 2020) (DIKHANBAYEVA, 2020).
- Capacidade em tempo real: É definida como a capacidade de coletar, transferir, analisar, monitorar e compartilhar dados em tempo real (HERMANN; PENTEK; OTTO, 2016). Com as tecnologias de *Big data* e análise de dados, o *status* dos componentes do sistema/planta podem ser facilmente acessados e

monitorados, esse ambiente acelera o processo de decisão dos responsáveis, garantindo assim a sustentabilidade, a manutenção e segurança dos equipamentos e das operações (DIKHANBAYEVA, 2020) (HABIB; CHIMSOM, 2019).

- **Virtualização:** É a capacidade de tornar processos/objetos do mundo físico em cópias virtuais (FALCÃO, 2019). Esse processo ocorre quando os CPS conseguem monitorar os processos físicos através do acesso aos dados em tempo real provenientes de sensores, *Virtual reality* – Realidade Virtual (VR), e *Argumented Reality* – Realidade Argumentada (AR) (PAELKE, 2014). Além das vantagens no gerenciamento da planta, a virtualização oferece a possibilidade de testar e aprimorar o design de protótipos (MATSAS; VOSNIAKOS; BATRAS, 2018) simulando previamente no ambiente virtual para serem posteriormente implementados no ambiente real (FALCÃO, 2019).
- **Modularidade:** É a capacidade que os sistemas modulares tem de se adaptarem em situações diferentes, tornando-se flexíveis a substituição e/ou expansão de outros módulos da rede (HERMANN; PENTEK; OTTO, 2016). Dessa forma, o sistema deve ser facilmente ajustável conforme a necessidade, desde as alterações de produtos a implementação de novas tecnologias (CARVALHO; CAZARINI, 2020).
- **Descentralização e decisões Autônomas:** É a capacidade dos CPS de tomarem decisões próprias (FALCÃO, 2019). As decisões são automatizadas e baseadas na coleta e na análise dos dados vindos de sensores e pela identificação por *Radio Frequency Identification* – Identificação por Rádio Frequência (RFID), que rastreiam os dispositivos conectados. Assim, é possível alterar o grau de “liberdade” de decisões do sistema conforme o processo e da hierarquia da organização (DIKHANBAYEVA, 2020).
- **Orientação a Serviços:** É a capacidade de integração de serviços das empresas e dos CPS com os humanos, utilizando da IoS como meio (HERMANN; PENTEK; OTTO, 2016). Esses serviços podem ser utilizados interna ou externamente na indústria (DIKHANBAYEVA, 2020) tornando assim possível uma inclusão ágil dos clientes com o produto e processo, aumentando o valor e a rentabilidade final para a indústria (HABIB; CHIMSOM, 2019).

Segundo (RÜSSMANN, 2016) os nove pilares tecnológicos da indústria 4.0: Integração Horizontal e Vertical, IoT, *Additive Manufacturing* – Manufatura Aditiva

(AM), *Autonomous Mobile Robots* – Robôs Móveis Autônomos (AMR), *Big data* e análise de dados (*Big Data Analytics*), *Clouding Computer* – Computação em Nuvem (CC), simulação e ciber segurança.

2.2 SISTEMAS CIBER-FÍSICOS

Segundo (LEE; SESHIA, 2016) o termo CPS foi utilizado pela primeira vez por Helen Gill em 2006, para referir-se à integração da computação (objetos cibernéticos) com os processos físicos (objetos físicos). Entende-se por objetos cibernéticos qualquer *hardware* e/ou *software* que consegue realizar funções de computação, controle e comunicação em um ambiente discreto; os objetos físicos serão qualquer sistema natural, operados em um sistema contínuo (HU, 2014).

Essa integração de sistemas se dá pela comunicação e armazenamento de dados entre diferentes dispositivos conectados em uma rede, estes que interagem e trocam informações com o mundo físico por meio de sensores e atuadores, tornando-se assim uma importante ferramenta para controlar e gerenciar o comportamento de produtos, dispositivos e serviços (JAZDI, 2014) (ESTERLE; GROSU, 2016). Esses conceitos, formam a tríade que constituem os CPS, conforme mostra a Figura 3.

Figura 3: Sistemas ciber-físicos.



Fonte: Indústria 4.0: conceitos e fundamentos, Sacomano (2018).

Os CPSs são uma nova geração de tecnologias que englobam outros conceitos bem conhecidos, como a IoT, Indústria.4.0, *Machine to Machine* – Máquina para Máquina (M2M), IoT, cidades inteligentes, entre outros (LEE; SESHIA, 2016)

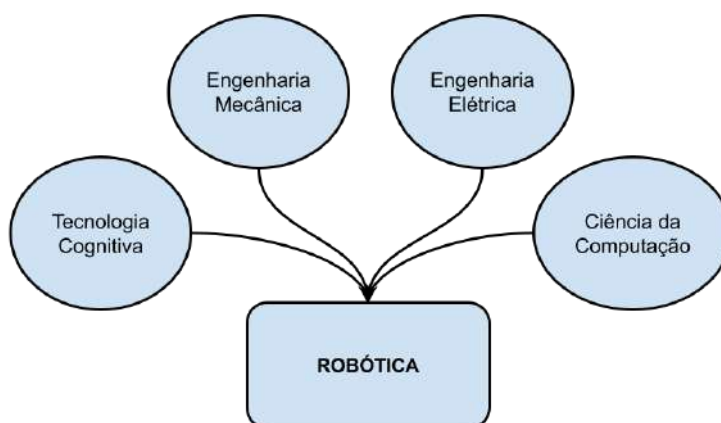
(ESTERLE; GROSU, 2016). Essas tecnologias, já presentes em nosso cotidiano, são aplicadas em diversos campos, como na área da saúde, transporte, cibersegurança, controle de infraestrutura crítica, eletrônica, sistemas de tráfego, manufatura e robótica distribuída (MORAES, 2013) (ANUMBA; AKANMU; MESSNER, 2010).

Segundo os autores Baheti e Gill (2019) e HU (2014) os CPSs serão um agente chave para o desenvolvimento de futuros tecnológicos e irão alterar como interagimos com o mundo físico. 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 (ESTERLE; GROSU, 2016).

2.3 ROBÓTICA MÓVEL

Por Matarić (2007), a Robótica é uma ciência, com sua origem na teoria de controle, cibernética e Inteligência Artificial (IA). Aplica o estudo do design, da manufatura, do controle de processos e da programação de robôs para resolver determinados problemas e/ou atender os humanos (MCKERROW, 1986). Dessa forma, o desenvolvimento interdisciplinar ocorre pelo conhecimento nos campos da engenharia mecânica, engenharia elétrica, tecnologia cognitiva e da ciência da computação (Figura 4) (SUGIHARA, 2022).

Figura 4: Áreas da Robótica.



Fonte: Adaptado de *Autonomous Robots and Vehicles*, Sugihara (2022).

Segundos estudos recentes pelos autores Yang (2018) a robótica enfrentará grandes desafios nos próximos 5 a 10 anos. São dez os obstáculos que envolverão a necessidade de novas descobertas, pesquisas e preocupações no nível de impacto socioeconômico: a fabricação de novos materiais, desenvolvimento de robôs bio-híbridos com inspiração biológica, novas fontes de energia, o desenvolvimento de enxames de robôs, a navegação e exploração em ambientes extremos, aspectos da *Artificial Intelligence* – Inteligência Artificial (AI) para robôs, o desenvolvimento de *Brain-computer Interface* – Interface Cérebro-Computador (BCI)s, interação social dos robôs com os humanos, robôs médicos e segurança e ética nas inovações na robótica.

- Robôs.

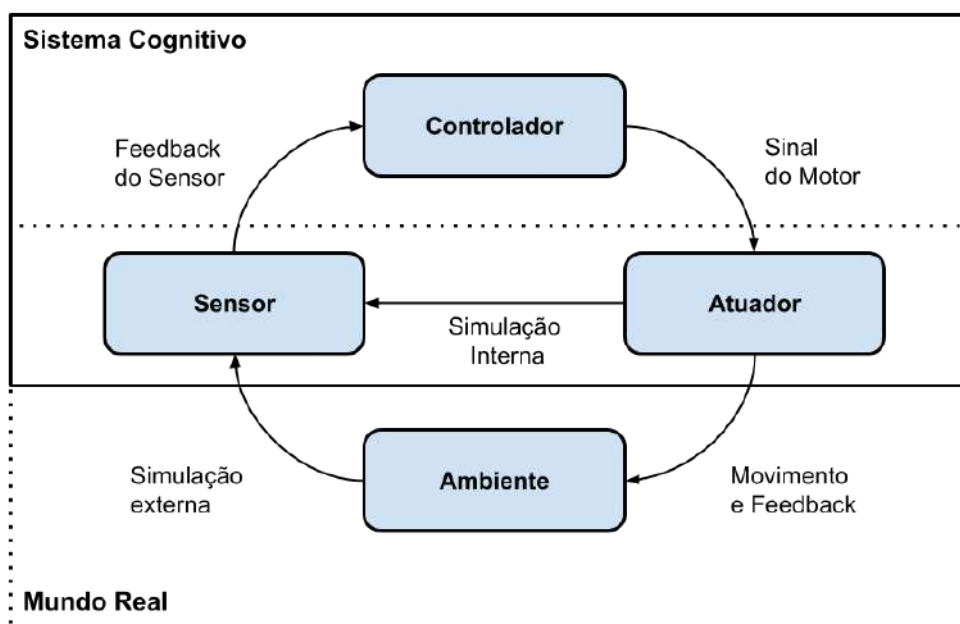
Um robô é um dispositivo reprogramável e manipulador multi-função capaz de se movimentar, manipular objetos, realizar uma determinada tarefa e interagir com o ambiente, dependendo do tipo de programação nele estabelecida (RUSSELL; NORVIG, 2003).

O sistema robótico é geralmente constituído de cinco subsistemas principais (CHEN, 2017) :

- Corpo/mecanismo e transmissores: sistemas físicos construídos com componentes/ou montagens que permitem a locomoção e transferência de forças no ao robô;
- Fonte de energia: fornece energia elétrica para os circuitos de controle e acionamentos acoplados ao robô;
- Atuador: conversor de uma energia elétrica em potência mecânica;
- Sensor: conversor de estímulos físico-químicos em sinais elétricos;
- Controlador: unidade de processamento que controla todos os componentes do robô, baseadas em um programa de *software*.

Dessa forma, o funcionamento desses subsistemas (corpo, fonte de energia, atuador, sensor e controlador) em harmonia pode ser correspondido ao um modelo de controle chamado acoplamento *loop sensorimotor*: processo contínuo que permite aos robôs interagir a um ambiente complexo (diferentes estímulos e grandezas físicas) quando os dados coletados pelos sensores são processados pelo controlador e reenviados para comandar os atuadores (MONTUFAR; GHAZI-ZAHEDI; AY, 2015), conforme a representação na Figura 5.

Figura 5: Fluxo *loop* sensorimotor.

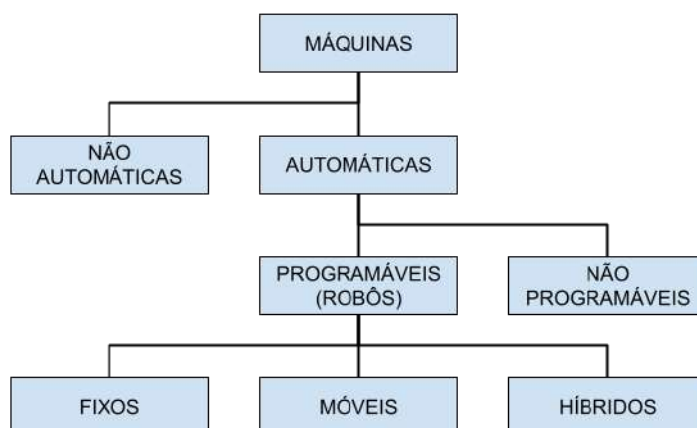


Fonte: Adaptado de *A Theory of Cheap Control in Embodied Systems*, Montufar, Ghazi-Zahedi e Ay (2015).

- Classificação de Robôs.

Na indústria operam diversas máquinas que substituem a mão-de-obra humana, mas não se caracterizam como robôs, diz Silva (2003). As máquinas são um sistema físico que utilizam de energia para aplicar forças e realizar atividades pré-definidas repetidamente, sendo estas automáticas ou não, necessitando da operação por um humano. Por outro lado, os robôs são uma combinação de máquinas, as quais são projetadas, podem ser reprogramadas, possuem inteligência, sensoriamento e um *software* avançado, conforme Nof (1999). Assim, esses robôs podem ser classificados como: fixos, móveis e híbridos, em relação às máquinas, conforme mostra 6.

Figura 6: Classificação de Robôs em relação às máquinas.



Fonte: Adaptado de Silva (2003).

- Robôs Móveis.

Os robôs móveis são um tipo de robô que se caracterizam por ser um sistema eletromecânico capaz de se mover autonomamente em um determinado ambiente, fala Tzafestas (2013). Por autônomo, entende-se um agente inteligente que possui um processo de raciocínio e aprendizado próprio — baseado em dados coletados do ambiente — decidindo assim seu próprio curso de ação ao invés de seguir instruções fixas, ou mesmo sendo controlados remotamente (SILVA ORTIGOZA, 2012).

Essa habilidade de deslocamento inteligente em um espaço livre dos robôs móveis se dá pelo processo de controle contínuo disposto na Figura 6. O mundo real é captado e traduzido pelo sensoramento, tornando-se dados que são enviados, interpretados e armazenados pelo *software*. Através do processamento desses dados, um modelo de mapa do ambiente pode ser gerado e o robô tem então a capacidade de se localizar no espaço. Com essas informações, o algoritmo introduzido no controlador do robô efetuará o planejamento e comandará as ações dos atuadores (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

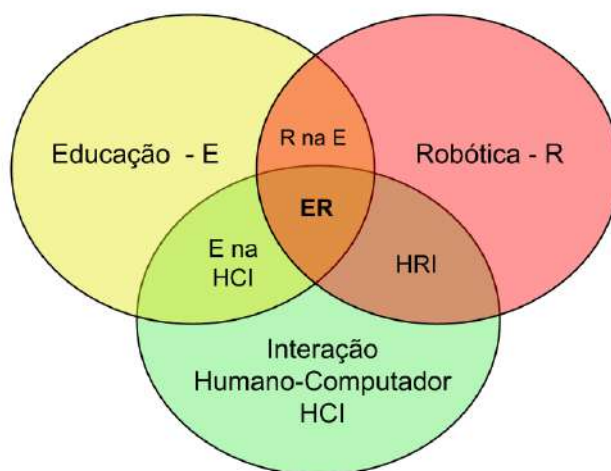
A arquitetura dos robôs móveis os torna uma ferramenta muito versátil, desenvolvidos para ambientes estruturados e não estruturados, para aplicações como as exploratórias: reconhecimento, inspeções, planetárias; as interventivas: manipulação de componentes perigosos, ataques terroristas; e as assistenciais: transporte de carga, medicamentos e alimentos, segurança, limpeza, reconhecimento de situações perigosas e operações militares (BRUZZONE; QUAGLIA, 2012) (SILVA, 2003).

- Robótica educacional.

Segundo Angel-Fernandez e Vincze (2018) a *Educational Robotics* – Robótica Educacional (ER) é uma área de estudo que visa principalmente melhorar a experiência de aprendizagem através do uso, do desenvolvimento e da validação de tecnologias que envolvem os robôs. Dessa forma, o uso da robótica nas escolas promove a construção de conhecimentos com base em um modelo construtivista ¹ fazendo a aprendizagem se tornar algo significativo para os próprios estudantes.

A ER é formada por três grandes áreas: educação (estudo e melhoria da aprendizagem); robótica (estudos e desenvolvimento de robôs); e *Human Computer Interaction* – Interação Homem-Computador (HCI) / estudo das interações humanas com os computadores a fim de melhorar a utilização (ANGEL-FERNANDEZ; VINCZE, 2018). Conforme mostra a Figura 7 as intersecções das áreas formam a Educação baseada em HCI, *Robotics in Education* – Robótica na Educação (R in E) e a *Human Interaction Robots* – Robôs de Interação Humana (HRI).

Figura 7: Conceito de Robótica Educacional.



Fonte: Adaptado de *Towards a formal definition of educational robotics*, Angel-Fernandez e Vincze (2018).

A implementação da robótica educacional nas escolas integra-se na era de mudanças na educação (alavancadas pela transformação digital e acesso à informação) com o uso das metodologias ativas como o *Project-Based Learning* – Aprendizagem Baseada em Projetos (PBL), *Lean Education Technology* – Tecnologia de Educação Enxuta (LET) e *Science, Technology, Engineering, Arts, and*

¹Nota de texto: “Segundo CASTANON (2015) PIAGET (1979) apresenta seu modelo construtivista de desenvolvimento cognitivo sustentado por dados empíricos, com um sujeito construtor, através da ação no mundo, de suas próprias estruturas cognitivas [...]”

Mathematics – Ciência, Tecnologia, Engenharia, Artes e Matemática (STEAM) Margrin (2022). Logo, em um ambiente de aprendizagem ativa os alunos assumem um papel central (possuem engajamento e constroem o conhecimento em torno de problemas reais) e os professores tornam-se mediadores e facilitadores no processo de aprendizagem (PISCHETOLA; DE MIRANDA, 2019).

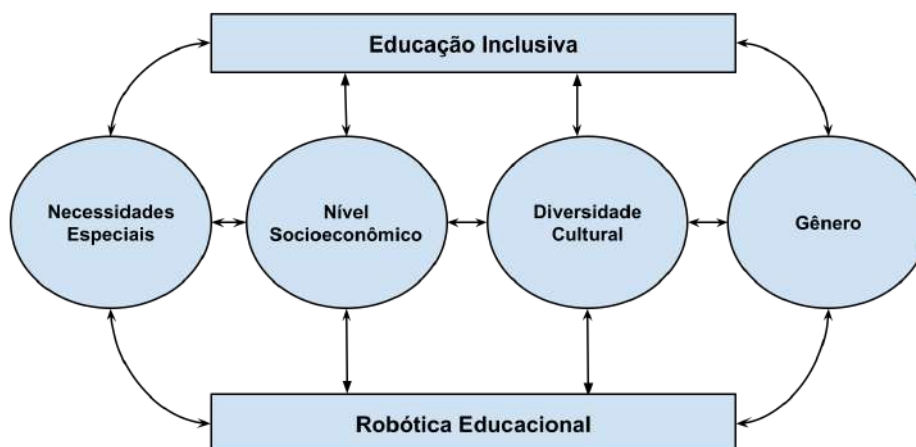
Assim, é possível listar as principais vantagens da utilização da robótica móvel nas escolas (ALIMISIS, 2013): o acesso ao conhecimento nas áreas da ciência, matemática, física, computacional, informática e engenharia, que constituem os robôs; o desenvolvimento pessoal dos alunos, em suas propriedades cognitivas (atenção, raciocínio, memória, imaginação) e meta-cognitivas (automonitoramento pessoal); *social skills* (construção de relacionamentos) e *soft skills* (conjunto de habilidades e competências relacionadas ao comportamento humano); e a possibilidade de criar experiências de aprendizagem construcionista para crianças.

- Robótica educacional e Educação inclusiva.

A Educação inclusiva promove o direito fundamental a todos de terem acesso à educação independente das diversidades étnicas, culturais, sociais, intelectuais, físicas, de gênero e sensoriais dos seres humanos (STUBBS, 2008). Conforme mostra a Figura 8 A ER também pode ser utilizada como uma ferramenta importante quando vista sob a dimensão de quatro níveis que constituem a educação inclusiva (DANIELA; LYTRAS, 2018): estudantes com necessidades especiais, de níveis socioeconômico, de diferentes culturas e de diferentes gêneros sexuais. Em relação à educação inclusiva utilizando a robótica educacional, o foco deste projeto se dá pelo nível de impacto socioeconômico:

- Nível Socioeconômico: dificuldades financeiras influenciam no acesso à educação (desde o custo para manter as necessidades básicas como as relacionadas ao ambiente escolar). O uso da ER possibilita a esses estudantes acesso a uma qualidade de ensino superior, com metodologias e materiais diferenciados, antes possíveis apenas em instituições particulares.

Figura 8: Relações entre Robótica educacional e Educação inclusiva.



Fonte: Adaptado de *Educational robotics for inclusive education*, Daniela e Lytras (2018) e

- Robôs Educacionais.

Os robôs educacionais são robôs projetados para serem utilizados tanto no ambiente escolar como no doméstico, tendo como característica serem robustos e com um custo relativamente baixo (BEN-ARI; MONDADA, 2018). Os principais modelos de robôs educacionais podem ser divididos em: pré-montados, kits robóticos, kits de construção e robôs sociais. Para este projeto, considera-se como foco a análise dos kits robóticos.

- Kits Robóticos: São kits de construção para montagem de robôs, geralmente personalizáveis e usados principalmente em protótipos para projetos e no ensino de robótica (VIRNES, 2014). Esses kits podem ser do tipo de: construção corporal (permitem criação de esqueleto mecânico), kits de *software* (permitem testes e desenvolvimento de *firmwares* para simuladores), kits programáveis (não possibilitam modificações de hardware, direcionamento para a reprogramação do *firmware*) e *Starter Kits* (fornecem diversos recursos para modificação corporal, eletrônica, mecânica e de *software*). Entretanto, costumam ser mais caros que os robôs pré-montados (RUZZENENTE, 2012). Exemplos de kits robóticos conforme a Figura 9 (da esquerda para a direita): LEGO Mindstorm EV3, Turtlebot, Sparki e Mbot.

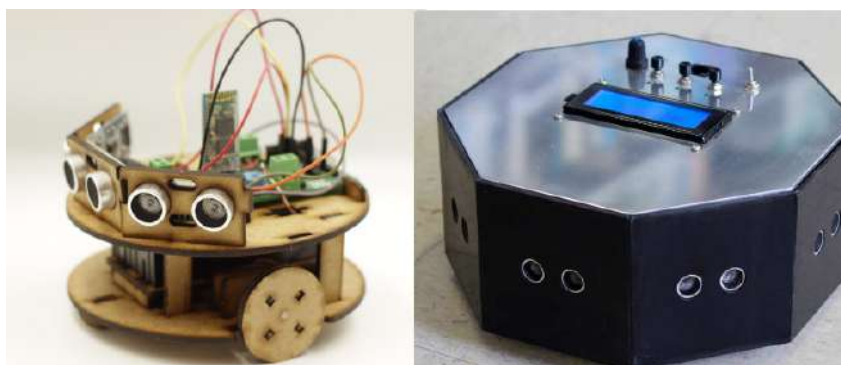
Figura 9: Kits Robóticos educacionais.



Fonte: Adaptado de Lego *Mindstorms Education EV3*, (LEGO, 2022); *Sparki — Programmable Arduino STEM Robot Kit for Kids*, (ARCBOTICS, 2022); *TurtleBot3* (TURLEBOT, 2022); e *mBot Robot Kit — Bluetooth Version — by Makeblock* (ADAFRUIT, 2022).

Ainda outros exemplos de robôs educacionais desenvolvidos no meio universitário, os robôs móveis ROME (Plataforma desenvolvida por Magrin (2022)), sendo projetado como uma ferramenta de baixo custo e enquadrada da educação inclusiva; e robô OCTO, plataforma desenvolvida Deda e Magrin (2020), projetada para a aplicação dos conceitos de robótica e posteriormente utilizada como uma aplicação de um DT (MAGRIN; BRITO; TODT, 2019).

Figura 10: Robô Móveis educacionais: ROME (Esquerdo) e OCTO (Direito).

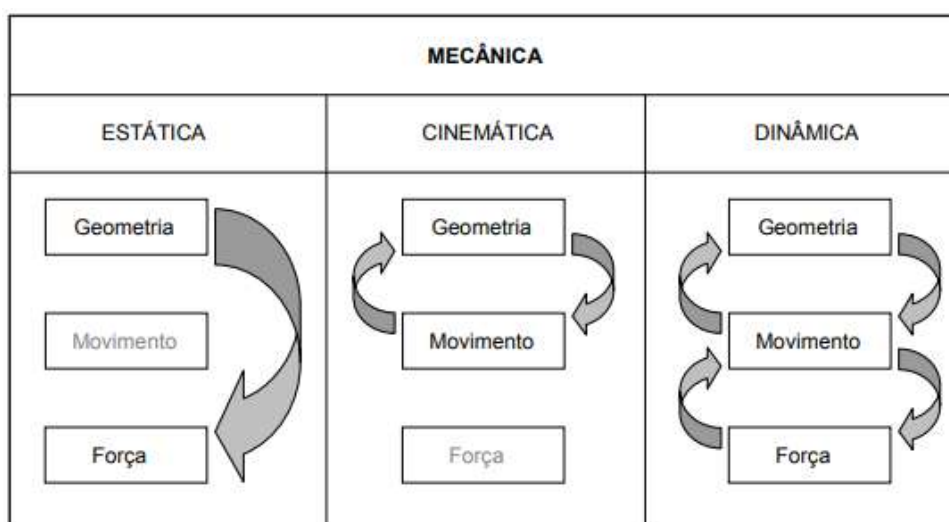


Fonte: Adaptado de Magrin (2022) e Magrin, Brito e Todt (2019).

2.3.1 Sistema Mecânico

Quando se projeta ou analisa um sistema mecânico, deve dividir-se o problema em duas partes distintas. Primeiramente, as dimensões de cada componente e a suas ligações devem permitir que possa determinar seu movimento. Em segundo lugar, cada um dos componentes deve conseguir resistir às forças que nele atuam. Desta forma, o estudo da cinemática e a dinâmica, desempenham um papel fundamental no estudo dos mecanismos robóticos. Esse estudo, segundo a mecânica clássica, pode dividir-se em três grandes disciplinas, a estática, a cinemática e a dinâmica, conforme mostrado na Figura 11 (FLORES; CLARO, 2005).

Figura 11: Disciplinas que constituem a mecânica: estática, cinemática e dinâmica.



Fonte: Adaptado de Flores e Claro (2005).

- Estática.

É o estudo das leis de composição das forças e as condições de equilíbrio dos corpos materiais submetidos à ação de forças e/ou momentos. O conhecimento de tais condições permite verificar a estabilidade das estruturas. Também é válida e aplicável quando se analisa mecanismos de baixas velocidades e acelerações (FLORES; CLARO, 2005).

- Cinemática.

Já a cinemática (do grego, movimento), é a área da Mecânica que se ocupa das leis do movimento dos corpos independentemente das forças que nele atuam. Neste tipo de análise apenas se estudam os aspectos puramente geométricos do movimento, não sendo considerados os esforços envolvidos no processo. Definir

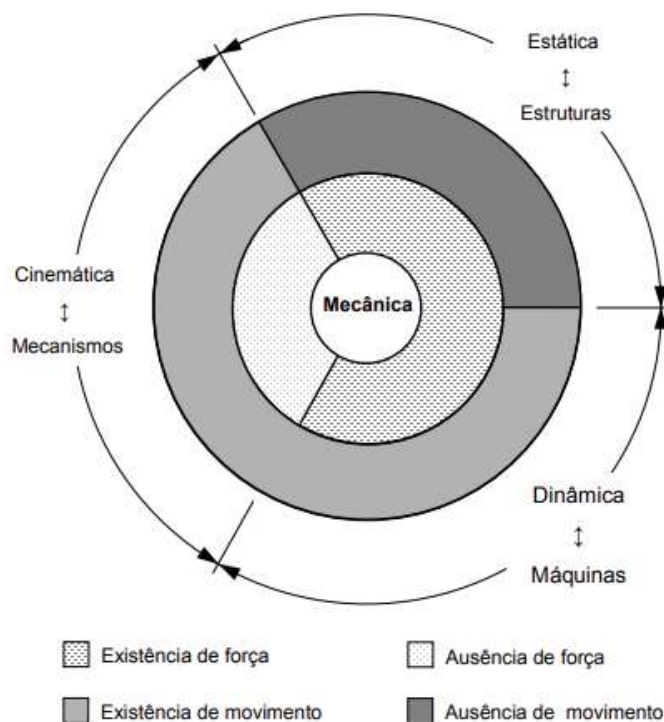
cinematicamente um mecanismo é determinar, a posição, a velocidade e a aceleração, para cada instante em relação a um referencial previamente definido (FLORES; CLARO, 2005).

- Dinâmica.

Por outro lado, a cinemática serve de base à dinâmica, uma vez que o estabelecimento das relações cinemáticas é necessário ao estudo do movimento dos corpos submetidos à ação das forças que nele atuam. Na dinâmica, consideram-se, também, a sua inércia. Com isso, pode-se prever o movimento causado por determinadas ações ou vice-versa. Estas leis foram sistematizadas e formuladas pela primeira vez por Newton na sua obra "*Principia Mathematica Philosophiae Naturae*", publicada em 1687 (FLORES; CLARO, 2005).

A Figura 12 sintetiza, de forma gráfica, os conceitos mecânicos estudados da estática, da cinemática e da dinâmica relacionados com as estruturas, os mecanismos e as máquinas.

Figura 12: Relações dos sistemas mecânicos.



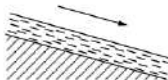


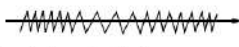






Fonte: Adaptado de Flores e Claro (2005).

Com o breve entendimento sobre os conceitos da Mecânica Clássica, desenvolvida por Newton, passa-se a observar estes conceitos aplicados à robótica móvel. Nesse sentido, o estudo da mecânica aplica-se à locomoção, estrutura e mecanismos de robôs móveis.

2.3.1.1 Sistemas de Locomoção

A automação, de modo geral, foi a grande percussora do desenvolvimento da robótica, seja ela móvel ou fixa². A necessidade de se ter uma máquina automática que pudesse interagir com o ambiente fez com que pesquisadores, cientistas e engenheiros se inspirassem em aspectos biológicos para isso. Dessa forma, sistemas robóticos de locomoção e manipulação foram baseados nisto. A seguir, na Figura 13, nota-se como se dá a cinemática aplicada a sistemas biológicos.

Figura 13: Mecanismos de locomoção utilizados em sistemas biológicos.

Tipo de movimento	Resistência ao movimento	Cinemática básica do movimento
Fluxo em um canal 	Força Hidrodinâmica	Tubilhões 
Rastejar 	Força de atrito	Vibração Longitudinal 
Deslizar 	Força de atrito	Vibração Transversal 
Correr 	Perda de energia cinemática	saltos periódicos em uma mola 
Caminhar 	Perda de energia cinemática	rolagem de um polígono 

Fonte: Adaptado de Siegwart, Nourbakhsh e Scaramuzza (2011).

- Pontos-chave da locomoção na robótica.

A locomoção segue o princípio inverso da manipulação, pois o robô fixo manipula objetos no ambiente, transmitindo a força a eles. Já na locomoção, o ambiente é fixo e o robô se move transmitindo força ao ambiente. Em ambos os casos, o estudo de Sistemas Mecânicos são a base científica para analisar as características de interação dos mecanismos com o meio físico.

Portanto, de acordo com Siegwart, Nourbakhsh e Scaramuzza (2011), locomoção e manipulação compartilham as mesmas questões centrais, conforme Tabela 1, de estabilidade, características de contato e tipo de ambiente:

²Nota de texto: “Robótica fixa são máquinas manipuladoras, com vários graus de liberdade, controladas automaticamente, reprogramáveis, multifuncionais, que podem ter a base fixa ou móvel, para utilização em aplicações de automação industrial”

Tabela 1: Pontos-chave para Locomoção.

Estabilidade	<ul style="list-style-type: none"> — Número e geometria dos pontos de contato; — Centro de gravidade; — Estabilidade estática/dinâmica; — Inclinação do terreno.
Características de contato	<ul style="list-style-type: none"> — Tamanho e forma do ponto de contato/caminho; — Ângulo de contato; — Fricção.
Tipo de ambiente	<ul style="list-style-type: none"> — Estrutura; — Meio físico (Terra, Água e Ar).

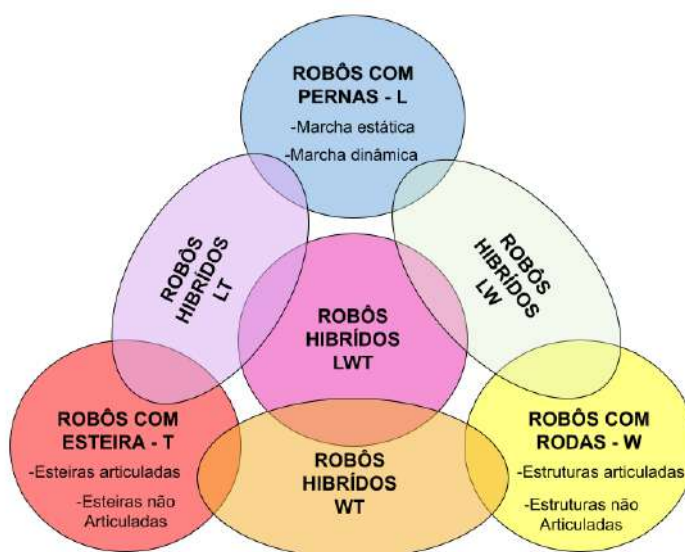
Fonte: Adaptado de Siegwart, Nourbakhsh e Scaramuzza (2011).

Tendo esses pontos-chave em mente, pode-se projetar os mecanismos de locomoção conforme o tipo de locomoção desejada. A seguir será demonstrado como é possível determinar essa questão.

2.3.1.2 Locomoção para Robôs móveis terrestres

Segundo os autores Bruzzone e Quaglia (2012) há três tipos principais de locomoção para os robôs móveis terrestres: *Legged Locomotion* – Locomoção por Pernas (L), *Wheeled Locomotion* – Locomoção por Rodas (W) e *Tracked Locomotion* – Locomoção por Esteiras (T). A combinação desses modelos resulta nos chamados robôs híbridos, sendo quatro categoriais: *Leg-Track* – Perna-Esteira (LT), *Leg-Wheel* – Perna-Roda (LW), *Wheel-Track* – Esteira-Roda (WT) e *Leg-Wheel-Track* – Perna-Roda-Esteira (LWT) (Figura 14). Essa ampla diversidade de locomoção é resultado de pesquisas e estudos por acadêmicos, a fim de obter uma combinação de novas características para esses robôs, como: acréscimo de velocidade, capacidade de ultrapassar obstáculos, capacidade de subir degraus e escadas, capacidade de subir declives e capacidade de adaptação em diferentes condições nos ambientes (ambientes estruturados e não estruturados).

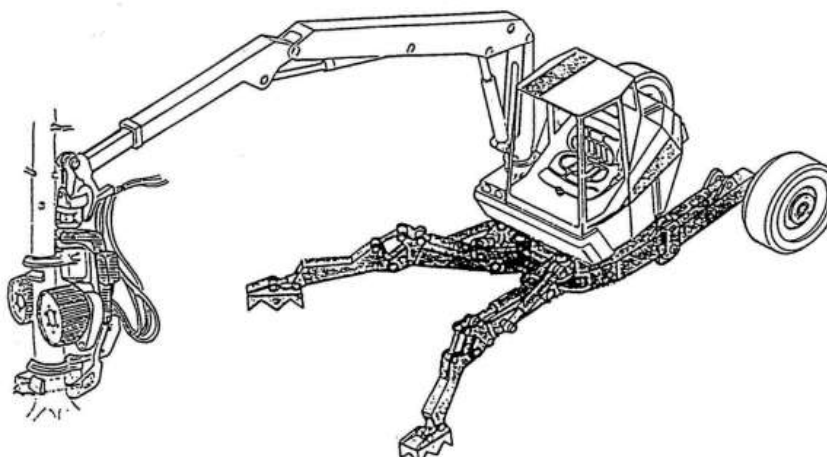
Figura 14: Categorias de Robôs móveis terrestres.



Fonte: Adaptado de *Review article: locomotion systems for ground mobile robots in unstructured environments*, Bruzzone e Quaglia (2012).

Na Figura 15 é possível verificar como ocorre um sistema de locomoção híbrida em um robô. Neste caso, o RoboTrac é aplicado a ambientes externos, como a extração de árvores em florestas, permitindo a fácil movimentação nos terrenos acidentados.

Figura 15: Exemplo de Robô Híbrido com sistema LW.



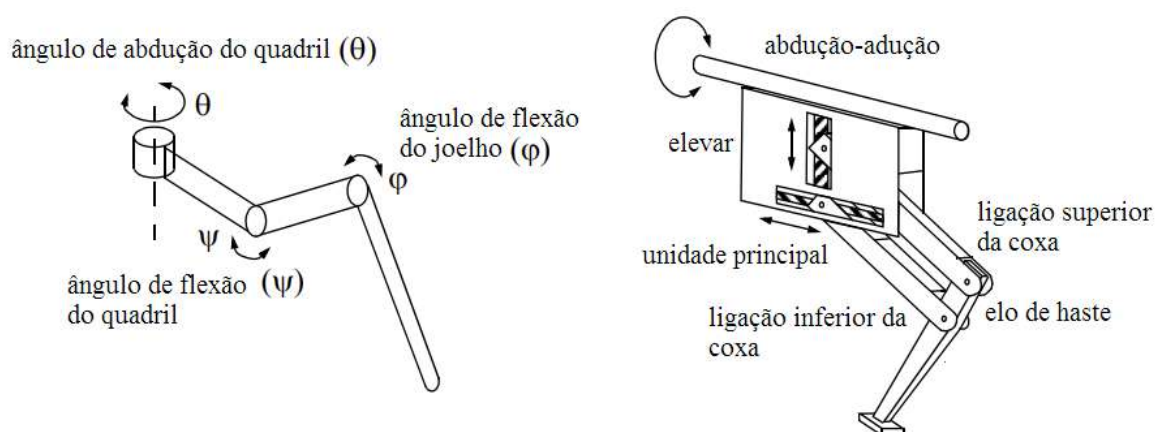
Fonte: Adaptado de Siegwart, Nourbakhsh e Scaramuzza (2011).

- Locomoção robótica — L.

São robôs dotados de um conjunto de pernas para se locomoverem, biologicamente inspirados em animais e caracterizados por pontos de contato com o chão.

Não são necessárias todas as pernas para manter o robô estável, sendo que por este fato se adaptam melhor em terrenos desestruturados e rudes, desdobrando-se nas habilidades de adaptabilidade, manobrabilidade e manipulação de objetos com destreza. Devido à locomoção por pernas ser de uma complexidade maior, em vista de necessitarem partes articuladas que requerem certo nível de liberdade para prover movimentos variados (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Figura 16: Exemplos de pernas mecânicas com três graus de liberdade.

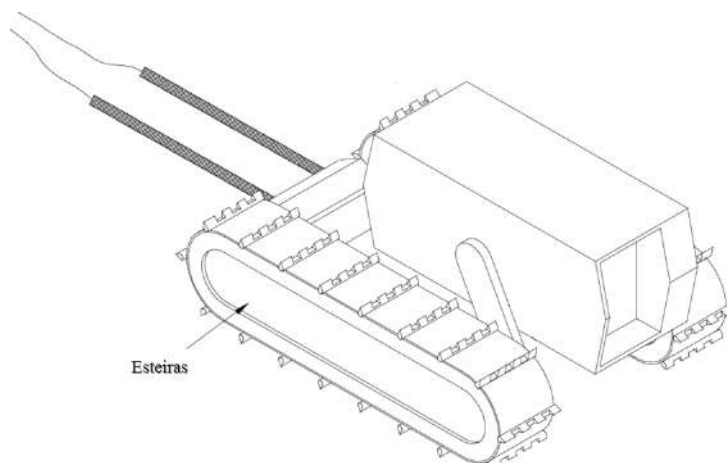


Fonte: Adaptado de Siegwart, Nourbakhsh e Scaramuzza (2011).

- Locomoção robótica — T.

Os robôs que usam esteiras têm áreas de contato com o solo muito maiores, são mais robustos e isso pode melhorar significativamente sua dirigibilidade em terrenos acidentados em comparação com projetos convencionais com rodas. O tanque do exército opera dessa maneira e o Nanokhod (Figura 17) é um exemplo de robô móvel baseado neste conceito. Entretanto, ao alterar a orientação do robô, devido a essa grande área de contato com o solo, esse sistema de locomoção derrapa com certa facilidade, dificultando a predição da posição e orientação exata (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Figura 17: Nanokhod — Exemplo de robô com esteiras.

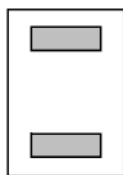


Fonte: Adaptado de Siegwart, Nourbakhsh e Scaramuzza (2011).

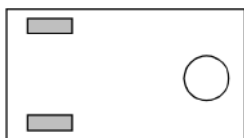
- Locomoção robótica — W.

São robôs que utilizam uma ou mais rodas para se locomoverem, podendo ser de projeto e construção simples devido a questões de custo e engenharia. Isso pode ser provado pelo simples fato de não ser necessário maior preocupação com equilíbrio nesses robôs, já que passam todo tempo em contato com o chão. O foco no estudo e projeto de robôs com rodas passa para problemas como, tração, estabilidade, manobrabilidade e controle. A Tabela 2, exibe alguns tipos de montagem de rodas (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

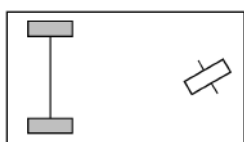
Tabela 2: Exemplos de sistemas com rodas.



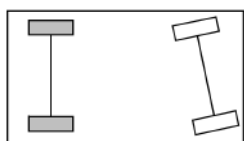
— Acionamento diferencial de duas rodas com o centro de massa abaixo do eixo. Ex.: conceito do Robô Tupy;



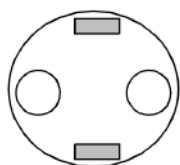
— Duas rodas acionadas independentemente, uma roda omnidirecional não motorizada na frente/traseira. Ex.: robôs educacionais;



— Duas rodas de tração conectadas (diferencial) atrás, uma roda livre dirigida na frente. Ex.: sistema aplicado em triciclos;



— Duas rodas motorizadas na traseira, duas rodas direcionais na frente. Ex.: carros;



— Acionamento diferencial de duas rodas com dois pontos adicionais de contato. Ex.: robôs educacionais;

Fonte: Adaptada de Siegwart, Nourbakhsh e Scaramuzza (2011).

2.3.1.3 Cinemática do Robô Móvel

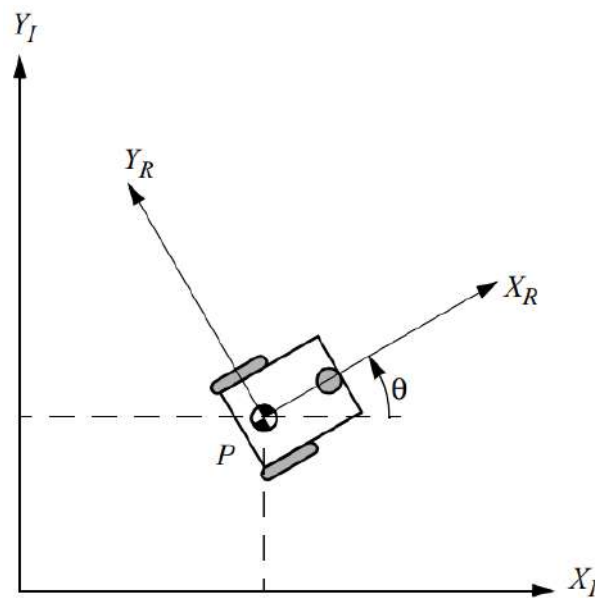
Na robótica móvel, é preciso entender o comportamento mecânico do robô. Isso é necessário para projetar um *hardware* que melhor atenda as características do ambiente. Por outro lado, é apropriado entender como implementar o *software* para ser possível controlá-lo com precisão. Portanto, o estudo da cinemática é fundamental para este contexto. Vale lembrar que para robôs móveis de alta veloci-

dade, as restrições dinâmicas também devem ser expressas e consideradas devido à inércia³ (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

- Representação do robô no plano.

Na Figura 18, é possível notar como o robô é representado no plano cartesiano de coordenadas X_I e Y_I . Nota-se também que o robô possui suas próprias coordenadas X_R e Y_R , baseadas no centro de giro P . Desta forma, é possível definir um ângulo θ do robô em relação ao plano principal.

Figura 18: Plano de referência global e plano de referência local do robô.



Fonte: Adaptado de Siegwart, Nourbakhsh e Scaramuzza (2011).

De acordo com Siegwart, Nourbakhsh e Scaramuzza (2011), pode-se descrever a posição como um vetor ξ_I baseado nesses três elementos, com a seguinte matriz 2.1:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \quad (2.1)$$

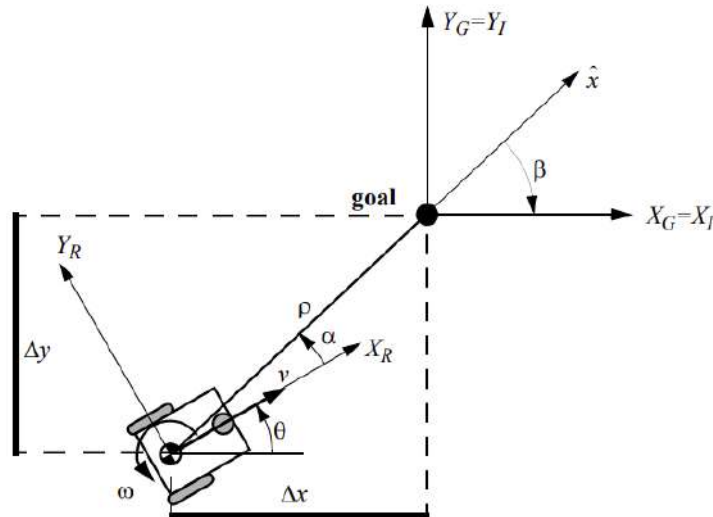
- Conceito de cinemática aplicada robótica.

A literatura de Siegwart, Nourbakhsh e Scaramuzza (2011), traz a Figura 19, como exemplo de um robô com uma posição e orientação arbitrárias e uma alvo

³Nota de texto: “Inércia é a propriedade de um corpo que faz com que ele se oponha a qualquer agente que tente colocá-lo em movimento ou, caso se encontre em movimento, altere a magnitude ou a direção de sua velocidade. Um corpo em movimento continua movendo-se não por sua inércia, mas devido à ausência de uma força capaz de retardá-lo, mudar sua direção ou acelerá-lo.”

predefinido. Com isso, é possível expressar os elementos de interesse, como, por exemplo, a orientação, a posição, ângulo e velocidade.

Figura 19: Cinemática de robôs e seus elementos de interesse.



Fonte: Adaptado de Siegwart, Nourbakhsh e Scaramuzza (2011).

- Modelagem cinemática.

A cinemática de um robô móvel de acionamento diferencial descrito no referencial inercial $\{X_I, Y_I, \theta\}$ é dada por:

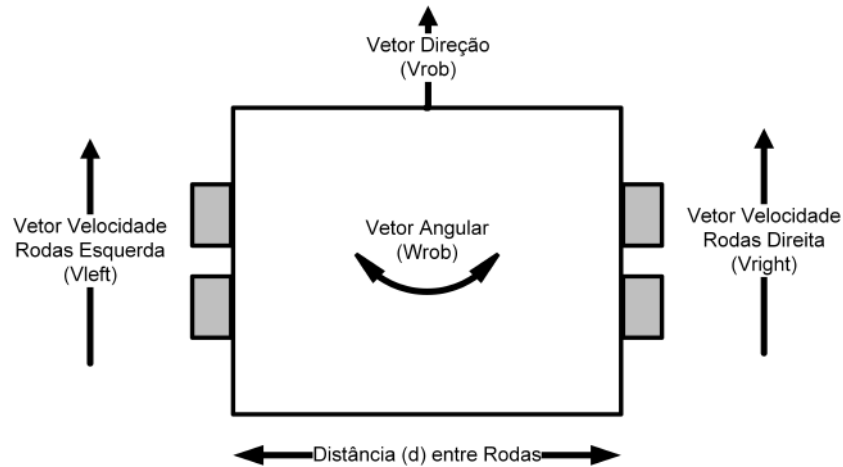
$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (2.2)$$

onde \dot{x} e \dot{y} são as velocidades lineares/angulares na direção do X_I e Y_I do referencial inercial.

- Cinemática do Robô Tupy.

Aplicando os conceitos previamente vistos, pode-se modelar a cinemática do Robô Tupy, arquitetando uma simplificação, as duas rodas juntas como uma (Figura 20):

Figura 20: Cinemática do Robô Tupy.



Fonte: Autor (2022).

onde é possível definir as equações da velocidade linear/angular para ambos os lados, sendo

$$v_{\text{Right}} = v_{\text{Rob}} + (d/2) \cdot \omega_{\text{Rob}} \quad (2.3)$$

$$v_{\text{Left}} = v_{\text{Rob}} - (d/2) \cdot \omega_{\text{Rob}} \quad (2.4)$$

$$\omega_{\text{Rob}} = (v_{\text{Right}} - v_{\text{Left}}) \cdot d \quad (2.5)$$

2.3.1.4 Materiais e Métodos de fabricação

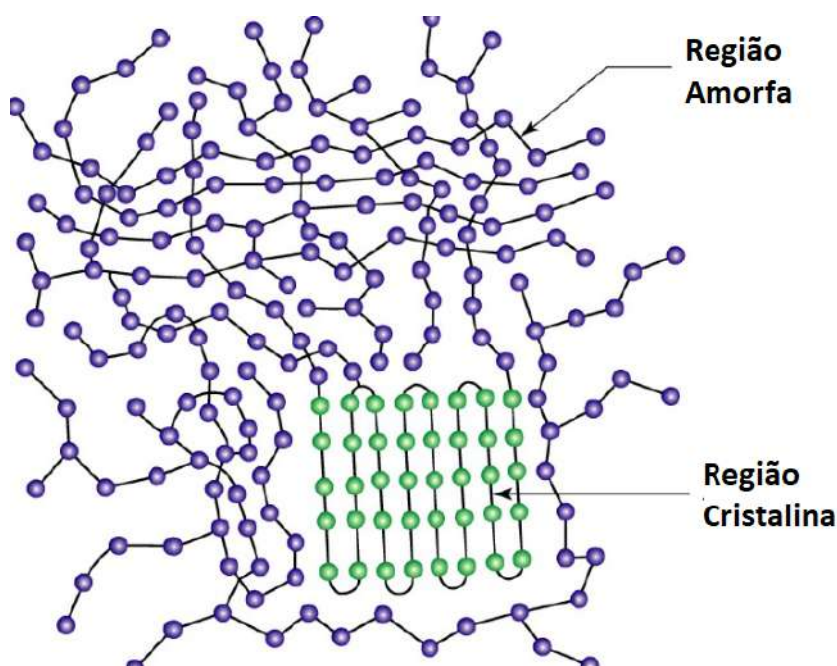
Os métodos de fabricação e materiais deste projeto se baseiam em conceitos intrínsecos a indústria 4.0, como os plásticos, matéria de prima de baixo custo e acessível para os processos de fabricação desenvolvidos; e a manufatura aditiva, pela versatilidade de prototipagem, custo-benefício e autonomia de design (BANDYOPADHYAY; BOSER, 2019)

- Plásticos.

Os plásticos são uma classe de material cuja constituição básica é a polimérica (material puro resultado do processo de polimerização de um longa cadeia de moléculas sintéticas chamadas monômeros) e a orgânica natural, descrita por R. Crawford (2020). Sendo considerado o material revolucionário do final do século XX, possui vantagens como: resiliência, resistência a corrosão, transparência, facilidade de processamento e leveza. São largamente utilizados em vários produtos

modernos, em áreas como as das embalagens, têxteis e eletrônica de consumo, como diz R. Crawford (2020). Os principais tipos de plásticos utilizados segundo R. Crawford (2020) são os termoplásticos, termofixos, compósitos e elastômeros. Para este projeto, é utilizado o polímero ácido polilático *Polylactic acid* – Ácido Polilático (PLA) como matéria-prima para a impressora 3D, dessa forma o foco é sobre os Termoplásticos.

Figura 21: Exemplo estrutural de um Termoplástico semicristalino.



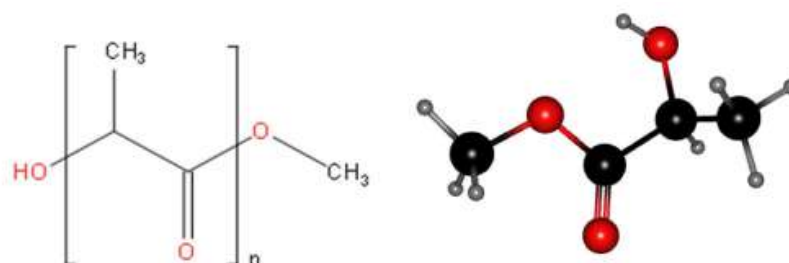
Fonte: Adaptado de *Characterization of Thermosets Part 21: Tensile Testing of Polymers; A Molecular Interpretation*, Gotro (2017).

- a. Termoplásticos: possuem longas cadeiras de moléculas (Figura 21) os quais são mantidas unidas por forças de Van der Waals relativamente fracas, desta forma quando o material é aquecido essas forças são enfraquecidas, alterando sua estrutura química, tornando-o flexível e re-moldável. Em relação a sua estrutura química, pode ser dividida em dois subgrupos: cristalina (ordenada): ponto de fusão alto, normalmente opaco, alta retração, alta resistência química, resistência a fadiga e ao desgaste. Exemplos: *Polietilo* – Polietileno (PE), *Polypropylene* – Polipropileno (PP), *Polyamide* – Poliamida (PA), *Polyesters* – Poliéster (PES), *Politetrafluoretileno* – Politetrafluoretileno (PTFE), PLA. E amorfa (aleatória): diferentes temperaturas para fusão, transparência, pouca retração, baixa resistência química, a fagia e ao desgaste. Exemplos: *Polyvinyl* – Polivinil (PVC), *Polystyrene* – Poliestireno (PS), *Polycarbonate*

– Policarbonato (PC), *Poly Methyl Methacrylate* – Polimetilmetacrilato (PMMA) e *Polyphenol Oxidase* – Polifenol Oxidase (PPO).

O PLA é um polímero termoplástico semicristalino mais importante do grupo de plásticos biodegradáveis, podendo ser produzido de forma natural ou sintética. Esse material é dito como sustentável, pois é neutro na utilização de carbono, com o uso de fonte renováveis como o amido de milho para sua composição. Suas principais aplicações são embalagem de alimentos (devido a sua biodegradabilidade), aplicações biomédicas (devido à biocompatibilidade com tecidos vivos) (AVÉROUS, 2008) e matéria-prima para impressoras 3D (R. CRAWFORD, 2020).

Figura 22: Composição Química do Ácido Polilático PLA.



Fonte: Adaptado de Martinez (2022).

/

As propriedades desse polímero dependem de suas características moleculares (Figura 22) da presença de estruturas ordenadas (cristalinidade, espessura cristalina, tamanho de esferulite, morfologia e grau de orientação em cadeia) (AVÉROUS, 2008). Dessa forma, possui excelentes propriedades mecânicas, químicas e térmicas, boa processabilidade, baixo nível de fusão, (HAMAD, 2015) rigidez, resistência a tração e permeabilidade a gases (MARTIN; AVÉROUS, 2001).

- **Manufatura Aditiva.**

Também conhecida como Impressão 3D, *Rapid Prototyping* – Prototipagem Rápida (RP), *Layered Manufacturing* – Fabricação em Camadas (LM) *Solid Free-form Fabrication* – Fabricação Sólida de Forma Livre (SFF), a AM é uma tecnologia que possibilita a construção por camadas livremente em 2 ou 3 dimensões, de qualquer peça ou protótipo baseadas em um desenho CAD (BANDYOPADHYAY; BOSER, 2019). Essas camadas são finas secções transversais da peça derivada do desenho CAD, sendo que, quanto mais fina for a camada, mais próxima à peça física será do modelo virtual (GIBSON; ROSEN; STUCKER, 2015).

A Figura 23 mostra um processo de fabricação genérico AM que envolve pelo menos oito passos, que podem mudar dependendo da complexidade do produto desejado e do método utilizado (GIBSON; ROSEN; STUCKER, 2015).

Figura 23: Etapas de um processo genérico de Manufatura Aditiva.



Fonte: Adaptado de *Additive Manufacturing Technologies 3D Printing, Rapid Prototyping, and Direct Digital Manufacturing*, Gibson, Rosen e Stucker (2015).

1. Modelo CAD: consiste em construir o modelo da peça com as dimensões externas em qualquer *software* de modelagem CAD;
2. Formato *Standard Template Library* – Biblioteca de Modelos Padrão (STL): o modelo CAD será convertido no arquivo padrão STL que descreve a peça em formatos de triângulos interligados;
3. Transferência do STL para a Máquina: o ficheiro STL deverá ser revisado (dimensões, orientação) e posteriormente introduzido na máquina de impressão;
4. *Setup* da Máquina: a máquina AM deve ser devidamente configurada antes do início do processo de construção. Os parâmetros da máquina são definidos conforme o projeto, seguindo restrições do material, fonte de energia, complexidade e outros;
5. Construção: parte automatizada do processo, apenas com a necessidade de supervisão da máquina, garantindo que não ocorram falhas no processo;

6. Remoção: com o processo concluído, é necessário aguardar o resfriamento da peça, tomando as devidas medidas de segurança;
7. Acabamento: as peças finalizadas necessitam de um pós-processamento (limpeza e inspeção);
8. Aplicação: peças podem ser utilizadas para as aplicações, montadas em conjunto com outras peças, ou ainda passada para outra etapa de tratamento (pintura e acabamento mais rigoroso)

A Tecnologia AM possui diferentes métodos de impressão que segundo Alghamdi (2021) podem ser classificados como baseados por metodologias de formação do produto, baseados pelo tipo de material usado e baseado no tipo de manufatura utilizada.

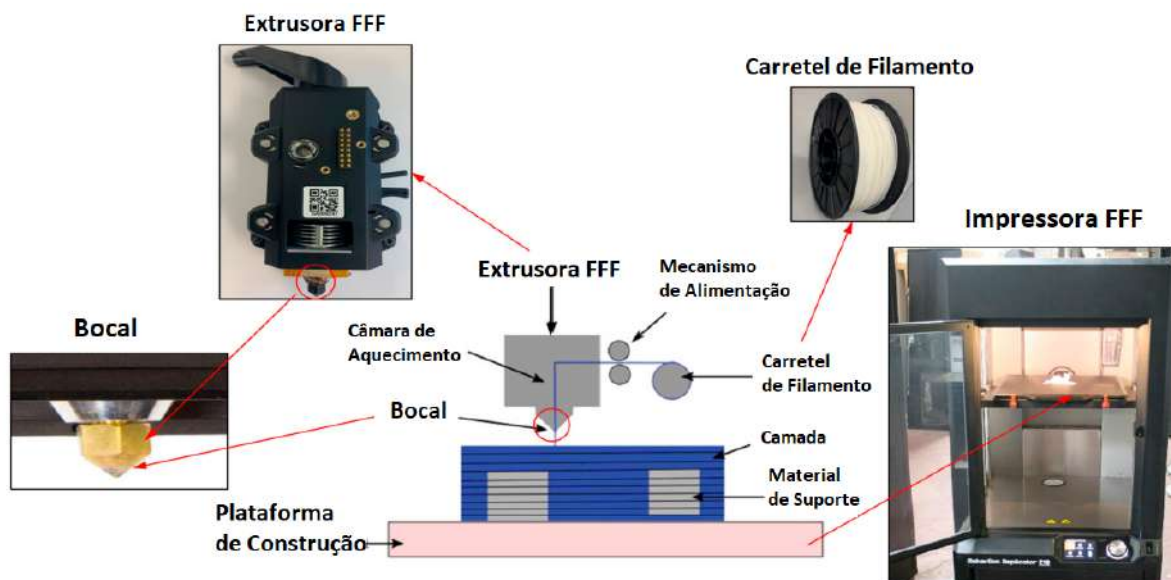
Para este projeto, o método de impressão de interesse é o da classificação baseados em tipo do material sólidos, a modelagem por deposição de fundidos *Fused deposition modeling* – Modelagem de Deposição Fundida (FDM).

O processo FDM também chamado de Fabricação por Fusão de Filamento *Field-flow fractionation* – Fracionamento de Fluxo de Campo (FFF) é um dos processos mais utilizados de manufatura aditiva, que utiliza do método de extrusão de termoplásticos em filamento para construir geometrias 3D complexas camada por camada (BRENKEN, 2018). Consiste em uma extrusora que possui dois bocais: uma é utilizada para modelar a peça e a outra para modelar o suporte da peça (R. CRAWFORD, 2020). A extrusora é aquecida (temperatura parametrizada), alimentada pela impressora com o filamento termoplástico (enrolado em um carretel) e ligada a um sistema de três eixos (x, y, z) para movimentar-se ao longo dos planos necessários para atingir as dimensões da peça a ser fabricada. Uma vez no ponto pré-definido no plano, começa a extrusão do filamento, pé e bocal, em finos fios, depositados sobre um suporte (material que fica sobre a base de construção da impressora). Estes fios formam ao longo do suporte uma camada (superfície ativada). Na sequência, a base de construção da impressora se movimenta para baixo (eixo z) para que a extrusora possa iniciar uma nova camada. A nova camada depositada irá colar na superfície ativada pela energia térmica armazenada na mesma. O processo se repete, até a peça estar finalizada (ARUP, 2021). A Figura 24 mostra um exemplo desse processo.

Ademais, o processo FDM/FFF é limitado para produção em larga escala por existirem certas restrições: matérias-primas limitadas, tamanho restrito das peças,

baixa taxa de escalabilidade, peças finais tem baixa qualidade e exigem acabamento (ARUP, 2021).

Figura 24: Exemplo de um sistema para fabricação por FDM/FFF.

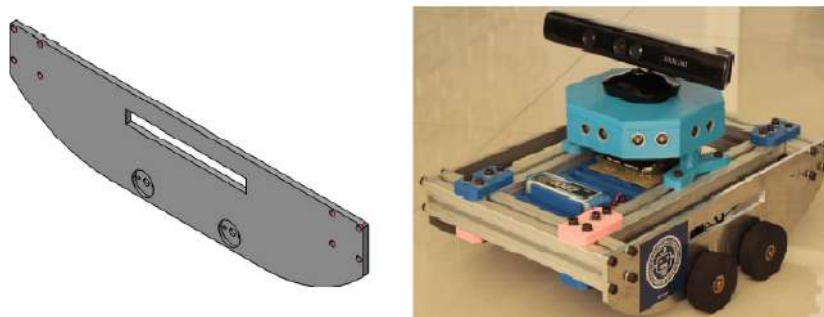


Fonte: Adaptado de *A review on filament materials for fused filament fabrication*, Arup (2021).

- Materiais e Métodos — Plataforma Mecânica robô Tupy.

A plataforma mecânica inicial do Robô Tupy (Figura 25) desenvolvida em *software Computer-Aided Three-Dimensional Interactive Application* – Aplicativo Interativo Tridimensional Auxiliado por Computador (CATIA) V5 *Student Edition* pelos autores Magrin, Del Conte e Todt (2021) utiliza métodos de fabricação baseada na tecnologia *Computer Numerical Control* – Controle Numérico Computadorizado (CNC) que consiste em programar instruções (baseadas em linguagem de código G) e parâmetros (posicionamento, avanço, velocidade) para serem executadas na máquina que irá fabricar a peça desejada de forma automática (THYER, 1991). A máquina utilizada nesses processos possui múltiplas ferramentas como fresas, tornos, lasers, soldas e montagem.

Figura 25: Robô TUPY-4WD versão 1, vista completa e vista em explosão robô TUPY-4WD.



Fonte: Adaptado de Magrin, Del Conte e Todt (2021).

Segundo Magrin, Del Conte e Todt (2021) a escolha do material de alumínio para perfil estrutural (utilizados na montagem do chassi) deve-se pela grande versatilidade estrutural e facilidade no processo de usinagem por CNC.

2.3.2 Sistema Eletrônico

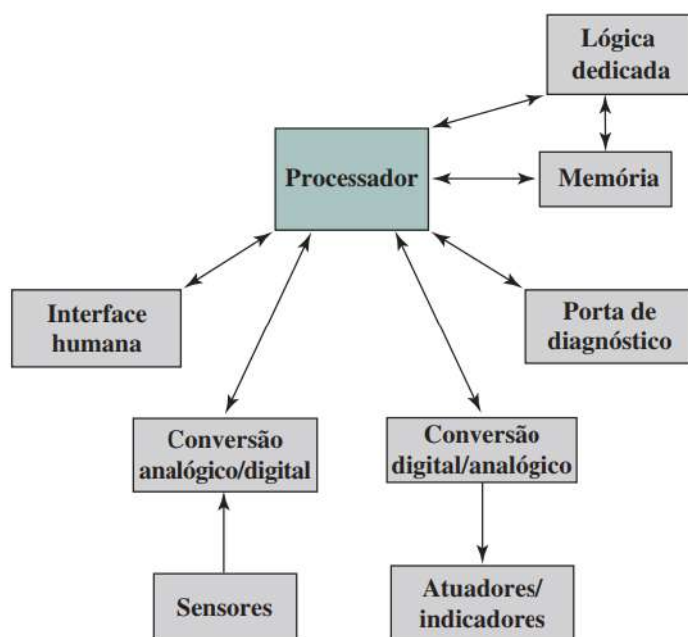
Na proposta da robótica móvel, baseado no contexto de sistemas CPS, a eletrônica tem papel fundamental, pois é dela que emana todo controle de periféricos e dispositivos através do uso de microcontroladores e microprocessadores. A eletrônica embarcada proporciona, a gestão do *hardware* de baixo nível, como a leitura de sensores e acionamento de atuadores. Já em um nível acima, o processamento de dados e interface interativa com o usuário, é dada pelo uso de microcomputadores. Com isso, pequenos robôs móveis, conseguem executar tarefas complexas com base no *feedback*⁴ da percepção do ambiente.

2.3.2.1 Sistemas Embarcados

Segundo Stallings (2017), o termo de sistemas embarcados refere-se a implementação do controle de uma tarefa específica, por circuitos eletrônicos acompanhados de uma programação dentro de um produto, ao passo que se difere de um computador de mesa de convencional de uso geral. Vale mencionar que a IoT tem contribuído massivamente para o uso de sistemas embarcados, devido seu conceito aplicado aos dispositivos inteligentes. Na Figura 26, nota-se a possível organização de um sistema embarcado.

⁴Nota de texto: “*Feedback*, em eletrônica, é um termo que descreve a comparação da saída real de um sistema com a saída desejada e o ajuste da atuação para produzir o resultado desejado”.

Figura 26: Elementos típicos de um sistema embarcado.



Fonte: Adaptado de Stallings (2017).

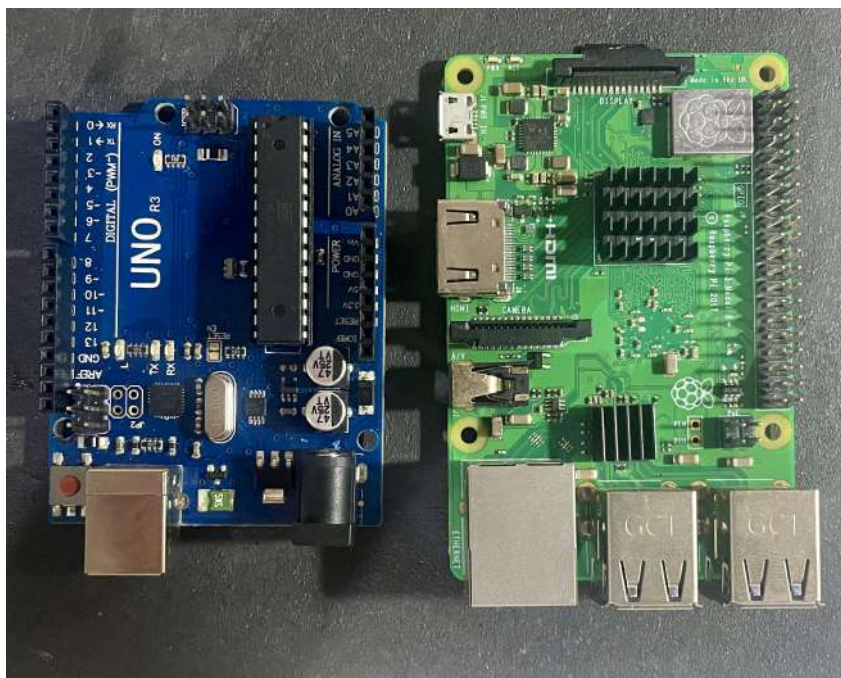
- Microcontroladores e Microprocessadores.

Os microcontroladores estão disponíveis em uma variedade de tamanhos físicos e capacidade de processamento. Podem ser programados para uma tarefa específica, embarcada em um dispositivo e executando tarefas conforme necessário (STALLINGS, 2017).

Numa abordagem voltada à robótica móvel, têm-se os microcontroladores ATmega328p 8-bits utilizados em plataformas como Arduino UNO R3, por exemplo e o microprocessador Cortex-A53 (ARMv8) 64-bits equipando o Raspberry Pi 3b+. Esses dois modelos são vastamente utilizados pelas comunidades acadêmicas, pesquisadoras e desenvolvedoras. A grande vantagem da utilização destas plataformas é a versatilidade da integração de dispositivos baixo nível (sensores e atuadores) com ambiente de alto nível (sistemas operacionais computacionais e robóticos), através dos protocolos de comunicação e de bibliotecas pré-compiladas.

Entretanto, um ponto a ser considerado, é que estes dispositivos servem, fundamentalmente, para prototipagem, não sendo recomendado para operações hostis e severas, como ambiente industrial. A Figura 27 mostra a plataforma Arduino UNO R3 à esquerda e o Raspberry Pi 3b+ à direita.

Figura 27: Arduino UNO R3 e Raspberry Pi 3b+.



Fonte: O Autor (2022).

2.3.2.2 Comunicação

Um dos elementos mais fundamentais de sistemas robóticos, indústria 4.0 e CPSs é a comunicação. Segundo Parede e GOMES (2011), protocolos de comunicação são um conjunto de regras definidas em que consta o formato no qual a mensagem deve ser transmitida entre os dispositivos conectados em rede. Para que essa rede de comunicação seja construída, é necessário um meio físico que permita a interligação desses dispositivos.

Com a necessidade da interligação entre redes de diferentes arquiteturas, a *International Organization for Standardization* – Organização Internacional para Padronização (ISO), com o intuito de padronizar os protocolos em camadas, estabeleceu o modelo *Open Systems Interconnection* – Interconexão de Sistemas Abertos (OSI)⁵.

Segundo Réu Junior (2010), o modelo OSI é dividido em sete camadas e cada camada é responsável por resolver um domínio específico de problemas de acordo com sua característica de transferência de dados. As camadas prestam serviços às camadas superiores, por meio de uma interface bem definida. Uma

⁵Nota de texto: “Modelo OSI: embora esteja praticamente em desuso, ainda serve como referência de estudo, pois oferece uma visão hierárquica do grupo de serviços que compõem cada camada, além dos protocolos e interface de comunicação entre elas”.

camada não interfere nas funcionalidades de outra e todas se comunicam por protocolos independentes, de forma que a eventual substituição de um protocolo em uma delas não influencia o funcionamento das demais, conforme a Figura 28.

Figura 28: Representação das camadas do modelo OSI e o método de comunicação



Fonte: Adaptado de Réu Junior (2010).

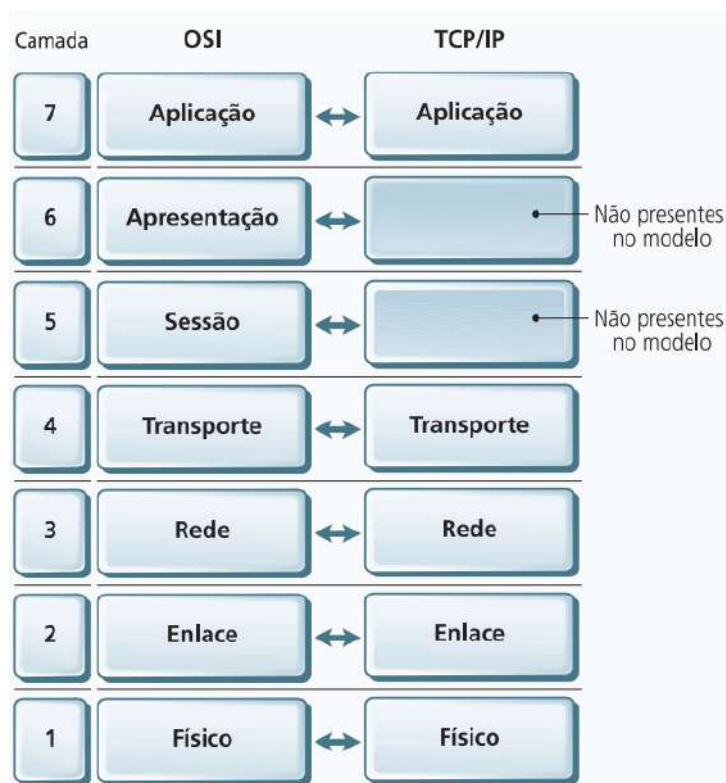
1. A camada física, determina como serão realizadas as transferências de bits através de um canal de comunicação, isto é, questões como tensão, duração dos sinais e meios físicos são tratadas neste nível.
2. A camada de enlace é responsável pela transformação dos dados brutos (bits) para uma transmissão enquadrada, aparente livre de erros;
3. A camada de rede controla o tráfego de dados, roteando pacotes da origem para serem encaminhados até o destino correto;
4. A camada de transporte tem como função básica aceitar os dados da camada superior (sessão), fragmentando essas informações à camada inferior (rede). Essa camada também determina qual tipo de serviço deve ser fornecido à sessão. O tipo de conexão de transporte mais comum é o modelo ponto a ponto;
5. A camada de sessão permite que usuários estabeleçam conexões entre si;

6. A camada de apresentação permite a comunicação entre computadores com diferentes representações internas dos dados, relacionadas as informações transmitidas.
7. A camada de aplicação é a interface com o usuário. Nela constam alguns protocolos de aplicação como:
 - *File Transfer Protocol* – Protocolo de Transferência de Arquivos (FTP) que permite transferências de arquivos entre computadores;
 - *Simple Mail Transfer Protocol* – Protocolo Simples de Transferência de Correio (SMTP) que gerencia a entrega das mensagens via correio eletrônico (e-mail);
 - *Simple Network Management Protocol* – Protocolo Simples de Gerenciamento de Redes (SNMP) que realiza o gerenciamento de rede;
 - *Secure Shell* – Protocolo de Rede Criptográfico (SSH) que proporciona a emulação de terminais remotos para acesso a outros computadores;
 - *Network File System* – Sistema de Arquivos de Rede (NFS) que possibilita a utilização de armazenamento remoto como local;
 - *Hypertext Transfer Protocol* – Protocolo de Transferência de Hipertexto (HTTP) que promove a comunicação entre navegador e servidores *web*;
 - *Domain Name System* – Sistema de Nome de Domínio (DNS) que mapeia o nome de um endereço numérico na rede.
- Protocolo TCP/IP.

Em redes, tem-se o *Transmission Control Protocol/Internet Protocol* – Protocolo de Controle de Transmissão/Protocolo de Internet (TCP/IP), baseado em modelo OSI, sendo um conjunto de protocolos que visa a interconexão entre dispositivos em uma rede. Esse protocolo é a base da comunicação moderna (redes locais e internet). Além disso, com os avanços dos CPSs e IoTs, seu uso tende a ser ampliando.

Na Figura 29, o protocolo TCP/IP abstrai as camadas de **apresentação** e **sessão** do modelo OSI, simplificando o protocolo. Na prática, essas camadas eram pouco utilizadas na implementação das aplicações e, caso necessárias, essas funções são incluídas na camada de aplicação.

Figura 29: Diferenças entre modelo OSI e TCP/IP



Fonte: Adaptado de Réu Junior (2010).

- Protocolo USB.

Atualmente, o método físico para troca de dados mais utilizado é a transmissão serial, ou seja, bit a bit, por meio de sinal elétrico ou óptico. A grande vantagem de enviar e receber informações serialmente é a redução do *hardware*, pois utiliza, primordialmente, o par *Receive* – Receber (RX)/*Transmit* – Transmitir (TX) e mais a referência *Ground* – Terra (GND), diferentemente da comunicação paralela. A Tabela 3 ilustra, genericamente, a comparação entre algumas comunicações seriais, lembrando que existem outras versões destes protocolos com velocidades e alcances diferentes;

Tabela 3: Comparativo entre exemplos de comunicação serial.

Interface	Tipo	Distância máxima (metros)	Velocidade máxima (bps)	Exemplos aplicados
USB 3.0	<i>dual simplex</i>	3	5G	Transferência de dados
USB 2.0	<i>half duplex</i>	5	1,5M, 12M, 480M	Teclado, mouse
Ethernet	serial	500	10G	Redes locais
I2C	síncrono	5,4	3,4M	Microcontroladores, relógio em tempo (RTC)
RS-232 (EIA/TIA-232)	assíncrono	30	20k	CLPs, IHMs, instrumentação
RS-485 (EIA/TIA-485)	assíncrono	1200	10M	Aquisição de dados e controle de sistemas

Fonte: Adaptado de (AXELSON, 2009)

Segundo Axelson (2009), os benefícios do *Universal Serial Bus* – Barramento Serial Universal (USB) são facilidade de uso, rapidez e confiabilidade, transferências de dados, baixo custo e economia de energia se comparada a outras interfaces, conforme visto os exemplos na Tabela 3.

Na robótica móvel beneficia-se disso, pois há uma ampla gama de dispositivos vem com essa tecnologia integrada, como, por exemplo, placas Arduino UNO e Raspberry Pi. Além disso, as portas USBs oferecem alimentação de 5V para seus dispositivos.

- Comunicação sem fio — *Wireless*.

Para robótica móvel, o uso a comunicação sem fio é essencial para troca de dados. Alguns protocolos foram estabelecidos pela IEEE, visando padronizar algumas arquiteturas físicas de comunicação.

- *Wireless Local Area Network* – Rede Local Sem Fio (WLAN): a IEEE-802.11 (2021), estabeleceu a arquitetura para tratar das redes *Wi-Fi*, estabelecendo alguns padrões, com objetivo de fornecer conectividade sem fio para estações fixas, portáteis e móveis em uma área local. Este padrão também oferece aos órgãos reguladores um meio de padronizar o

acesso a uma ou mais bandas de frequência para fins de comunicação de área local;

- *Wireless Personal Area Network* – Rede de Área Pessoal Sem Fio (WPAN): a IEEE-802.15 (2020), tratou de estabelecer o padrão de arquiteturas como *Bluetooth* e *ZigBee*⁶. O padrão define a camada física e as especificações de subcamada *Media Access Control* – Controle de Acesso de Mídia (MAC)⁷ para conectividade sem fio de baixa taxa de dados com dispositivos fixos, portáteis e móveis sem bateria ou requisitos de consumo de bateria muito limitados. O propósito visa especialmente os requisitos de comunicação para IoT.

2.3.2.3 Percepção

Segundo Sacomano (2018), sensores são dispositivos que respondem a estímulos de condições físico-químicas, como luminosidade, temperatura, pressão, corrente, aceleração, posição, entre outras, transformando-as em sinais elétricos que podem ser lidos e processados.

A percepção são os “cinco sentidos” da robótica, através dela é possível que o robô reconheça as interações com o ambiente real. A grande variedade de sensores utilizados, diferentes técnicas de medição para obter dados e usar diferentes interfaces para conectar a um controlador torna este assunto extremamente vasto.

Segundo Braunl (2022), do ponto de vista da engenharia, faz sentido classificar os sensores de acordo com seus sinais de saída para integrá-los com um sistema embarcado, por exemplo. A Tabela 4 mostra um resumo das saídas dos sensores com seus sensores típicos.

Tabela 4: Exemplos de sinais de sensores utilizados na robótica.

Sinal de Saída do Sensor	Exemplo de Sensor
Sinal Digital (0/1)	Sensor de Presença
Sinal Analógico (ex. 0 a 5V)	Acelerômetro
Sinal de Tempo (ex. PWM)	Sensor Ultrassônico
Serial Link (USB)	Câmera Kinect

Fonte: Autor (2022).

⁶Nota de texto: O ZigBee tem-se popularizado pela utilização em dispositivos IoT, como lâmpadas inteligentes.

⁷Nota de texto: MAC é um endereço físico e único, associado à interfaces de comunicação utilizadas em dispositivos de rede.

- Sensores Ultrassônicos.

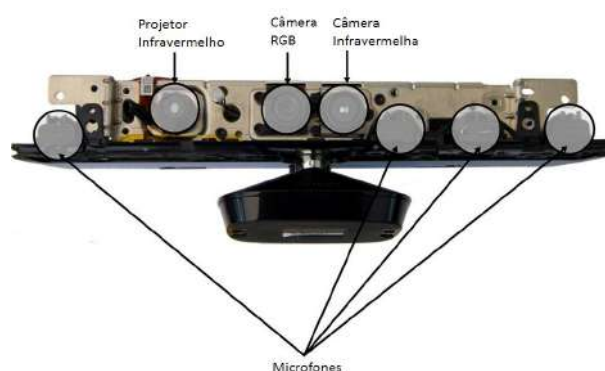
Sensores de distância ultrassônicos, são componentes eletrônicos baratos e simples. Eles têm uma vantagem sobre os sensores baseados em luz, pois o pulso de ultrassom resulta em um cone com um ângulo de abertura de 20 a 40 graus. Com isso, os sensores conseguem detectar pequenos obstáculos sem a necessidade de atingi-los diretamente com um raio de luz.

Segundo o Al Tahtawi (2018), o sensor funciona emitindo ondas ultrassônicas através do transmissor e então lê a reflexão da onda usando o receptor quando há um objeto detectado. Como exemplo, tem-se o sensor HC-SR04, onde a onda ultrassônica é emitida pelo transmissor através do pino **Trigger** com período de 10 μ s em uma frequência de 40 KHz. Então, a onda refletida recebido pelo receptor e encaminhado para o microcontrolador via pinos **Echo**.

- Sensores de Visão.

Sensores baseados em visão computacional são sistemas poderosos e complexos para captação de dados do ambiente para robôs. Há diversos tipos de câmeras que podem ser utilizadas para esta finalidade. Por exemplo, tem-se o Kinect Sensor, com um ótimo custo-benefício para aplicações na robótica móvel. Segundo El-laithy, Huang e Yeh (2012), o Kinect é uma câmera de profundidade usada na indústria de jogos para capturar movimentos de pessoas e jogadores eficientemente, usando a tecnologia de uma câmera *Red Green Blue* – Vermelho, Verde e Azul (RGB) e *Infrared* – Infravermelho (IR) para diferenciar a profundidade. Na Figura 30 verifica-se o *hardware* do Kinect.

Figura 30: Hardware Kinect.



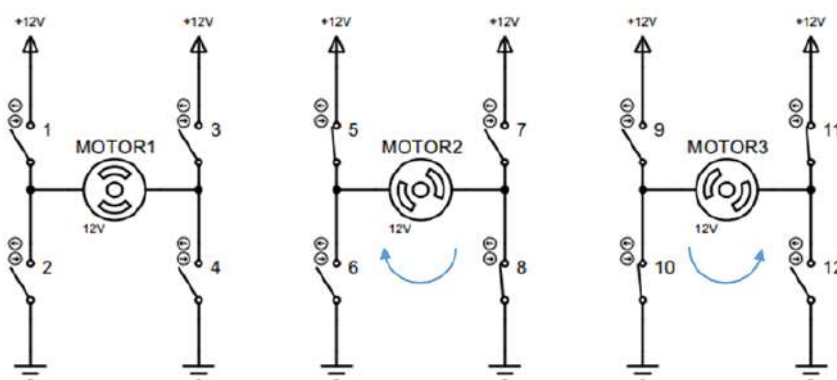
Fonte: Adaptado de Pearson (2012).

2.3.2.4 Periféricos de saída: módulos, acionamentos e alimentação

Para o robô se movimentar, seja por rodas, esteiras ou pernas, é necessário que haja dispositivos capazes de transformar sinais elétricos em movimento. Segundo Brauni (2022), motores elétricos, são o método mais comum usado para locomoção em robôs móveis. Todavia, em geral, os motores podem ser divididos em corrente alternada (CA) ou corrente contínua (CC), escovado ou sem escova, monofásico ou trifásico e motores especiais, como motores de passo ou servos.

A questão de controle dos motores, devido à necessidade de inversão no sentido de giro do motor para que o robô possa ir para frente e para trás, fica a cargo de placas eletrônicas de potência, também conhecidas como Ponte-H ou *drivers H-Bridge*, por exemplo, o circuito integrado L298N. Segundo Brauni (2022), a configuração da “Ponte H” recebeu esse nome por sua semelhança com a letra “H”, conforme diagrama da Figura 31.

Figura 31: Diagramas Ponte H com *software* Proteus.



Fonte: Autor (2022).

O funcionamento da Ponte H ocorre da seguinte forma:

- Condição do motor desligado: chaves (1, 2, 3 e 4) abertas;
- Condição do motor no sentido horário: chaves (5 e 8) fechadas e (6 e 7) abertas;
- Condição do motor no sentido anti horário: chaves (10 e 11) fechadas e (9 e 12) abertas.

Já questão da alimentação é fundamental para robótica móvel, pois estão sendo utilizados em diversas tarefas e em ambientes onde sua parada inesperada, pode gerar riscos, perdas e elevados custos. Atualmente, as baterias de Lítio têm

sido amplamente utilizadas, devido sua tecnologia de recarga, autonomia e baixa manutenção.

Já para conversão da tensão CC, elevação ou rebaixamento, pode se utilizar de circuitos conversores *buck*⁸ e/ou *boost*⁹ por conta da eficiência no processo de transformação, por exemplo, LM2596.

2.3.3 Sistemas Computacionais

Um sistema computacional moderno é composto por processadores, memórias, discos de armazenamento, interfaces de rede e periféricos de entradas e saídas (teclado, impressora, monitor), entre outros dispositivos. Essa gestão é feita via *software* através de um *Sistema Operacional* – Sistema Operacional (SO), fornecendo aos usuários, por programas, interface com o *hardware* simplificada. A Tabela 5 mostra a constituição de um sistema computacional por *hardware*, programas do sistema e programas da aplicação, do nível mais baixo ao mais alto, respectivamente (TANENBAUM; GONÇALVES; CONSULARO, 2003).

Tabela 5: Exemplo de composição de um sistema computacional.

Programas de aplicação	— Navegadores <i>web</i> — Simuladores — Ferramentas CAD/CAE/CAM
Programas do sistema	— Editores, Compiladores e Interpretador de comandos — Sistema Operacional
Camada Hardware	— Linguagem de máquina — Micro arquitetura — Dispositivos físicos

Fonte: Adaptado de Tanenbaum, Gonçalves e Consularo (2003)

⁸Nota de texto: um conversor *buck* (conversor abaixador) é um conversor de energia CC para CC que diminui a tensão (enquanto aumenta a corrente) de sua entrada (alimentação) para sua saída (carga).

⁹Nota de texto: um conversor *boost* (conversor elevador) é um conversor de energia CC para CC que aumenta a tensão (enquanto diminui a corrente) de sua entrada (alimentação) para sua saída (carga).

2.3.3.1 Sistemas Operacionais

Segundo Denardin e Barriquello (2019), o termo SO é um *software* ou conjunto de *softwares* cuja responsabilidade consiste na integração entre um sistema microprocessado e o usuário. De maneira geral, pode-se dizer que este sistema possui funções de gerenciamento de tarefas, memória e recursos. Nesse sentido, pode-se dizer que há duas formas distintas de conceituar e visualizar um sistema operacional, são elas:

- visão *top-down*: pela perspectiva do usuário ou projetista de aplicativos, provém a abstração do *hardware* do sistema microprocessado, intermediando o aplicativo e o *hardware*.
- visão *bottom-up*: opera como gerenciador, controlando a execução de tarefas e a utilização de recursos como protocolos de comunicação, *display*, entre outros.

No contexto geral, os SOs tem-se tornados cada vez mais difundidos. Em computadores, tem-se o exemplo de três principais sistemas como o Windows, Linux e MacOS. Já em dispositivos móveis, exemplo *smartphone* e *tablet*, os sistemas como Android e iOS são massivamente adotados.

Em sistemas embarcados, a implementação de sistemas dedicados também tem se tornando cada vez mais comum. Entretanto, há uma grande diferença, em relação aos sistemas operacionais multitarefas, pois nesse contexto, o sistema deve respeitar a premissa do processamento em tempo real.

Ainda, numa abordagem voltada a contextos específicos, tem-se, na robótica, o uso de sistemas para gerenciamento e integração de serviços e conexões como o ROS.

- Sistemas Operacionais Linux.

Em ambientes de desenvolvimento, principalmente, o Linux, assim como outros SOs, gerencia todos os recursos de *hardware* associados ao computador. A grande vantagem em utilizá-lo é a confiabilidade atrelada ao fato de ter custo zero de entrada, pois se trata de um *Open-Source Software* – Software de Código Aberto (OSS)¹⁰. Esse SO está incorporado em toda parte *smartphones* a carros,

¹⁰Nota de texto: “OSS é um software de computador lançado sob uma licença na qual o detentor dos direitos autorais concede aos usuários o direito de usar, estudar, alterar e distribuir o software e seu código-fonte para qualquer pessoa e para qualquer finalidade.”

supercomputadores e eletrodomésticos, *desktops* domésticos a servidores corporativos (LINUX, 2022).

O Linux tem diversas versões, tanto para aplicações de uso pessoal quanto para servidores. Essas versões são chamadas de distribuições (ou, na forma curta, “distros”). Quase todas as distribuições do Linux podem ser baixadas gratuitamente, gravadas em disco/*pen drive* USB e instaladas. As principais distribuições do sistema Linux são Debian e Ubuntu (LINUX, 2022).

Para a robótica móvel, o Linux pode ser utilizadas em microcomputadores, como o Raspberry Pi. Dessa forma, é possível tornar o robô como uma espécie de servidor, podendo acessá-lo remotamente.

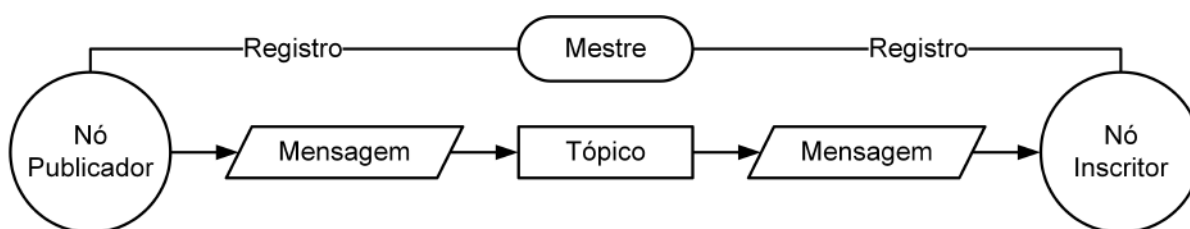
- Sistema Operacional em Tempo Real.

Em dispositivos embarcados, este tipo de sistema pode ser interessante, pois o processamento de dados, muita das vezes, deve ser realizado em tempo real. Segundo Denardin e Barriquello (2019), diferentemente de sistemas computacionais convencionais, o *Real Time Operating System* – Sistema Operacional em Tempo Real (RTOS) tem um propósito de realizar tarefas predefinidas, geralmente com requisitos específicos. Dessa forma, pode-se otimizar os recursos deste sistema, reduzindo tamanho, recursos computacionais e custo de produto/produção.

- Sistemas Operacionais Robóticos.

Semelhante aos SOs convencionais, devido à necessidade de gerenciamento e integração de hardware e software, tem-se, voltado principalmente à robótica móvel, o ROS é um conjunto de bibliotecas de *software* e ferramentas que ajudam a criar aplicativos voltados à robótica em código aberto. O conceito desta arquitetura pode ser descrita com a Figura 32 (ROS, 2022).

Figura 32: Conceito Arquitetura ROS

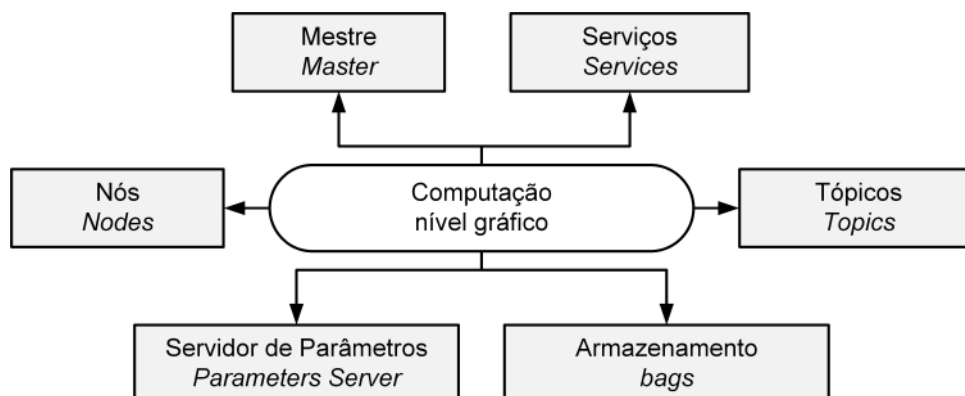


Fonte: Adaptado de ROS (2022).

Além disso, Martinez e Fernndez (2013), diz que o ROS cria uma rede onde todos os processos estão conectados. Qualquer nó do sistema pode acessar esta

rede, interagir com outros Nós, consultar as informações que estão sendo enviadas e transmitir os dados para a rede (Figura 33).

Figura 33: Arquitetura ROS nível gráfico computacional.



Fonte: Adaptado de Martinez e Fernandez (2013).

- Nós (*Nodes*) são processos onde a computação é feita. Os Nós, podem ser escritos com bibliotecas separadas do ROS em linguagem Python através do *rospy* ou em C++ com *roscpp*. Por exemplo, o processamento de dados de sensores, como os de visão computacional em tempo real, utiliza-se da biblioteca de código aberto OpenCV.
- Mestre (*Master*) o mestre, no ROS, fornece registro de nome DNS e pesquisa para o resto dos Nós, sem ele não haverá comunicação entre os Nós, serviços, mensagens e outros. Já no ROS2, versão em desenvolvimento, não há mais esse *Master*, ou seja, cada Nó consegue descobrir outros Nós independentemente, permitindo criar um sistema totalmente distribuído.
- Servidor de Parâmetros (*Parameter Server*) possibilita armazenar dados usando chaves em um local central. Com este parâmetro, é possível configurar os Nós durante a execução ou alterar o funcionamento deles.
- Mensagens (*messages*) são informações enviadas através conexão entre os Nós. Existem diversos tipos de mensagens padrões, por exemplo, (*geometry_msgs/msg/Twist.msg*).
- 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*. Já para os Nós receber Tópicos de outros Nós, simplesmente se inscreve no tópico de interesse com a função *Subscriber*. Exemplo (*/cmd_vel*).

- 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.
- Armazenamento (*bags*) permitem que seja armazenado e reproduzidos dados das mensagens. É uma ferramenta importante para o armazenamento dos dados, como dados de sensores. É comum utilizar isso no desenvolvimento de robôs complexos.

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 que, de acordo com Braunl (2022), são:

- Visualizador *rviz* — pode exibir graficamente a posição e a orientação de um robô em seu ambiente, com uma representação visual dos dados de seus sensores, por exemplo, *Light Detection and Ranging* – Detecção por Luz e Distanciamento (LiDAR)¹¹.
- Ferramentas de Interface *rqt* — pode ser usado para desenvolver interfaces de usuário específicas do ROS, por exemplo, utilização da ferramenta *rqt_graph* (Figura 34) para visualizar o gráfico de computação do ROS.
- Simulador Gazebo — é um simulador de robô 3D que agora é muito usado com o ROS.
- Modelagem URDF (*Unified Robot Description Format*) permite a definição de robôs móveis arbitrários e manipuladores de robôs. Muitos fabricantes de robôs comerciais fornecem arquivos URDF mais drivers de dispositivo ROS para seus sistemas de robôs.

Figura 34: Exemplo de rede ROS ativa com *rqt_graph*.



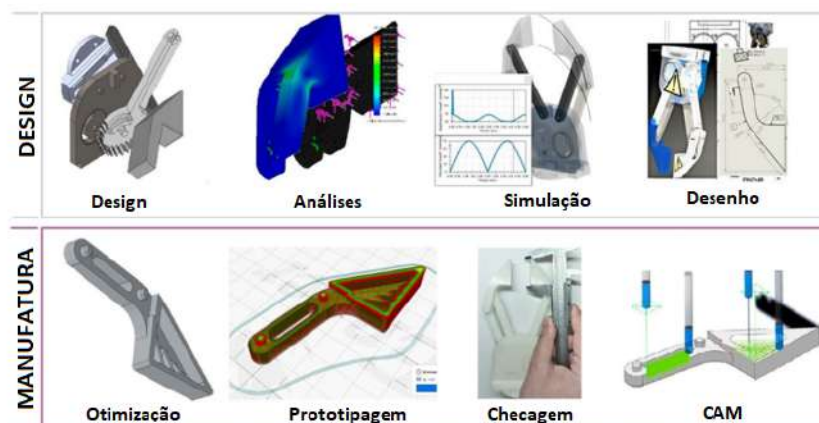
Fonte: Adaptado de DOCS.ROS (2022).

¹¹Nota de texto: LiDAR (*light detection and ranging*), é uma tecnologia óptica para determinar alcances (distância variável) visando um objeto ou uma superfície com um laser, através da verificação do tempo para a luz refletida retornar ao o receptor.

2.3.3.2 Modelagem Computacional 3D

As principais ferramentas computacionais para ser possível modelar o gêmeo virtual do robô TUPY, são as do tipo CAD (*Computer-Aided Design*), CAE (*Computer-Aided Engineering*) e CAM (*Computer-Aided Manufacturing*). A Figura 35 mostra um exemplo da utilização dessas ferramentas, com base na aprendizagem de metodologia ativa PBL (GUIRADO; VACAS; ALABANDA, 2022).

Figura 35: Utilização das Ferramentas CAD/CAE/CAM.



Fonte: Adaptado de Guirado, Vacas e Alabanda (2022).

O Desenho auxiliado por computador CAD possibilita o desenvolvimento de desenhos dos produtos, podendo ser em formato 2D ou 3D. As ferramentas CAD são baseadas na computação gráfica interativa. Segundo M.M.M. SARCAR K. MALLIKARJUNA RAO (2008) nesse sistema, o computador é utilizado como ferramenta de criação e manipulação de dados sob a forma de imagens/símbolos. Desta forma, as modificações que o engenheiro/utilizador faz no design do desenho são “traduzidas” em tempo real, transformadas em coordenadas, processadas e então reproduzidas na forma pictórica (HUMCKE, 1976).

A engenharia assistida por computador CAE possibilita diversas análises necessárias para a validação, desempenho e otimização do produto (LEE, 1999). Sendo algumas delas:

- Análise por elementos finitos *Finite Element Analysis* – Análise de Elementos Finitos (FEA): consiste em análises mecânicas simples (linear) e/ou avançadas (Não-linear). As análises simples fornecem características estruturais da peça, como aquelas relacionadas a resistência dos materiais (tensão, deformação, deslocamentos e factores de segurança) e de campos contínuos (calor, fluxo). Já as análises avançadas fornecem um resultado linear exis-

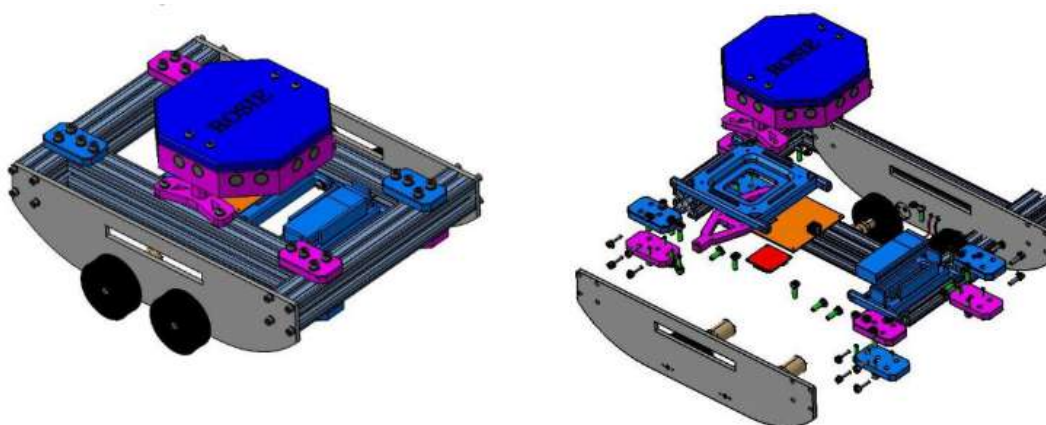
tente, ou representa de forma precisa os corpos com deformações excessivas (REDDY, 2005);

- Análise Cinemática: Para determinar movimentação e velocidades no mecanismo;
- Dinâmica dos fluidos computacional (*computational fluid dynamics — Computational Fluid Dynamics – Dinâmica de Fluidos Computacional (CFD)*): para simulação de fluidos (ESTORILIO; HATAKEYAMA, 1998) (WANG, 2017).

A manufatura assistida por computador CAM possibilita o controle das ferramentas de máquinas relacionadas ao processo e controle de manufatura do produto, operados diretamente ou indiretamente pelo computador. Um dos campos CAM mais conhecidos é o *numeric control* *Numeric Control – Controle Numérico (NC)*, usado para programar instruções de máquinas que realizar múltiplas funções (corte, perfuração, dobra, trituração) (LEE, 1999).

Um exemplo de uma modelagem CAD/CAE/CAM utilizando o *software* CATIA (Software também desenvolvido pela Dessault systems, é voltado para projetos que envolvem maior complexidade e que exigem alta qualidade, como projetos de infraestrutura, design de automóveis, aviões e navios (SYSTEMS, 2022)) é a modelagem virtual da primeira versão do robô Tupy, segundo Magrin, Del Conte e Todt (2021). Ver Figura 36.

Figura 36: Robô TUPY-4WD versão 1, vista completa e vista em explosão robô TUPY-4WD.



Fonte: Adaptado de Magrin, Del Conte e Todt (2021).

- Software de Modelagem Mecânica.

O SolidWorks, desenvolvido pela empresa Francesa Dassault Systemes S.A, utiliza do sistema de modelagem paramétrica 3D que possui funções de CAD, CAE

e CAM, onde a geometria é inicialmente desenhada em uma geometria 2D e posteriormente transformada em um modelo tridimensional. Para um modelo paramétrico, são considerados características como tamanho, forma, orientação e a localização, possibilitando a criação, a montagem e a simulação de sólidos e componentes virtuais (SHIH, 2014) e (HORTMAN, 2013).

Algumas das principais vantagens da utilização do SolidWorks: ambiente totalmente integrado, possui ferramentas que fornecem recursos de CAD, CAE e CAM no mesmo ambiente; Interface moderna e intuitiva, que permite um curva de aprendizado baixa; vasta comunidade online para suporte e tutoriais; ferramenta para *Product data management* – Gerenciamento de Dados do Produto (PDM), permitindo o controle do andamento do produto, como mudanças no projeto, lista de materiais, esquemas, ficheiros, de maneira simples (FIGUEIRA, 2002/2003).

- Software de Modelagem Eletrônica.

O Proteus Design Suite é um *software* para modelagem de projetos eletrônicos e automação desenvolvido pela Labcenter Electronics Ltd (LABCENTER, 2022).

Existem quatro módulos que trabalham em conjunto para fornecer as funcionalidades do *software* Proteus: Sistema de esquemático inteligente *Intelligent Schematic Input System* – Sistema de Entrada Esquemática Inteligente (ISIS), Sistema virtual de modelagem *Virtual System Modelling* – Modelagem de Sistema Virtual (VSM), simulador matemático PRO-SPIICE e Roteamento Avançado e Edição de *software Advanced Routing and Editing software* – *Software* Avançado de Roteamento e Edição (ARES) (ALNAHAM; SULIMAN, 2015).

Sendo o ISIS e VSM relevantes para este projeto:

- ISIS: para desenvolvimento de projetos de *multi-sheets*, relatórios e desenvolvimento de esquemáticos.
- VSM: desenvolvimento de simulações e animações em tempo real de componentes eletrônicos, usado paralelo ao ISIS. Através dessa tecnologia emulam-se microcontroladores por exemplo.
- Software de Simulação Computacional.

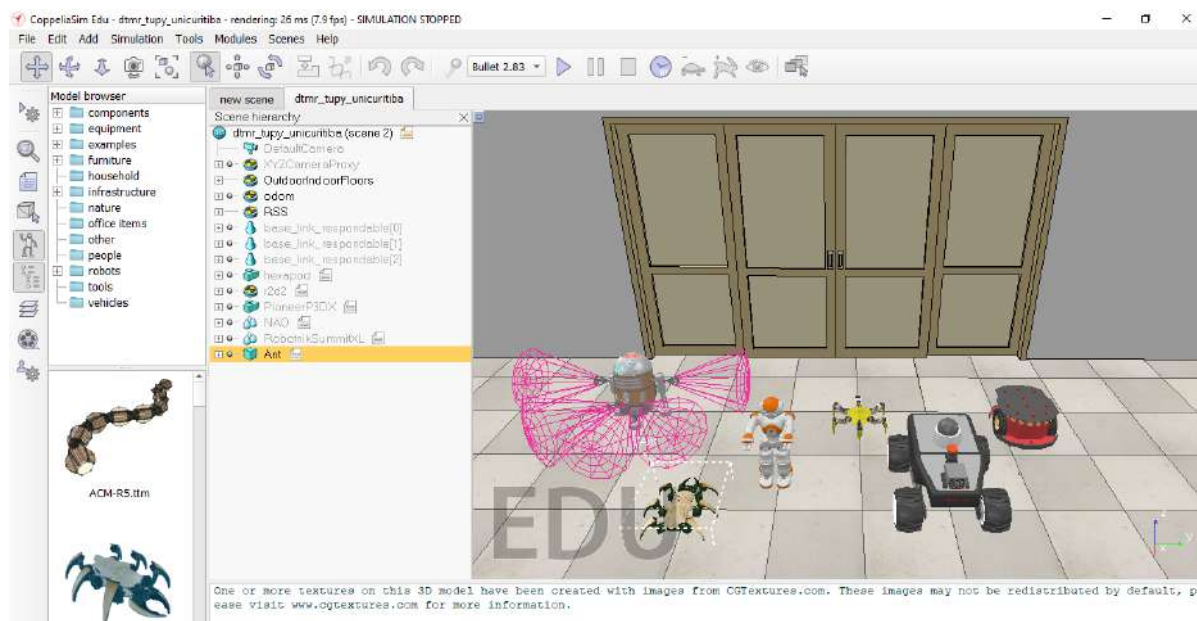
A simulação de uma cena real pode ser feita utilizando pacotes e bibliotecas cinemáticas, gráficas e físicas, de forma livre. Entretanto, segundo Rohmer, Singh e Freese (2013) nas aplicações de simulação que envolvem o uso de robôs, os

softwares devem ser mais robustos, possuindo uma arquitetura de controle estruturado para múltiplos modelos, controladores, de complexidade e facilmente codificados.

Nesse sentido, existem diferentes simuladores robóticos disponíveis, como o Open HRP (KANEHIRO, 2002), Gazebo (KOENIG; HOWARD, 2004), Webots (MICHEL, 2004) e CoppeliaSim (COPPELIA, 2022). Neste projeto o simulador robótico utilizado é o CoppeliaSim V 4.4 EDU, devido a sua flexibilidade de diferentes linguagens de programação, portabilidade para sistemas operacionais, poder de simulação, vínculo a protótipos, e principalmente por se tratar de um software gratuito.

CoppeliaSim (antigo *Virtual Robot Experimentation Platform* – Plataforma de Experimentação de Robôs Virtuais (V-REP)) é um *software* de simulação *framework open source* desenvolvido pela Coppelia Robotics AG (COPPELIA, 2022). Fornece diversas ferramentas para simulação em tempo real 3D de robôs comerciais e protótipos (Figura 37) (ROHMER; SINGH; FREESE, 2013). Sendo utilizado em aplicações como: monitoração remota, simulação de sistemas de automação de indústrias, controle de *hardware*, prototipagem e análise, controle de segurança, robótica educacional, diretamente para clientes e de outras de nível superior (SUNDANDHIRA VEERAN, 2021).

Figura 37: Exemplo de Simulação no CoppeliaSim.



Fonte: O Autor (2022).

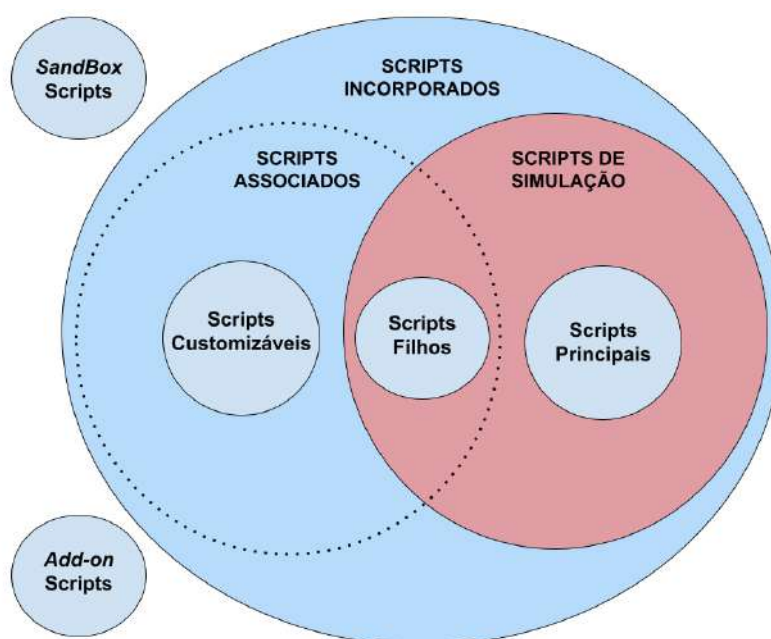
O CoppeliaSim possui uma arquitetura de controle distribuída (Figura 39) seu *framework* é composto por uma interface de programação de aplicações *Application Programming Interface* – Interface de Programação de Aplicativos (API) que possi-

bilita que um objeto ou modelo possa ser controlado individualmente por meio de scripts (bibliotecas compartilhadas).

Há dois grupos em relação ao tipo de API utilizada no CoppeliaSim (COPPELIA, 2022) (ROHMER; SINGH; FREESE, 2013): API Regular, formada por funções principais, nativas do CoppeliaSim. Tem em sua nomenclatura o prefixo *sim* com o ponto (exemplo: *sim.getObject*). Conforme mostra a Figura 38, os principais *scripts* desse grupo são:

- *Script* incorporado: escritos na linguagem nativa do CoppeliaSim (LUA ou Python). É chamado de *Script* principal (*main Script*) e faz parte diretamente de uma cena ou modelo. Esse tipo de *script* carrega nele próprio os *scripts* filhos (*child scripts*): coleção de rotinas para uma determinada função na simulação, associados aos objetos da cena; e os *scripts* customizáveis (*customization scripts*): coleção de funções para criar modelos personalizáveis na simulação, também são associados ao objeto da cena.
- *Sandbox script* e *add-ons script*: Também escritos em LUA ou Python, são *scripts* que permitem personalizar (fornecerem uma nova funcionalidade) objetos na simulação. Podem ser iniciados automaticamente ou em segundo plano.

Figura 38: *Scripts* nativos do CoppeliaSim.

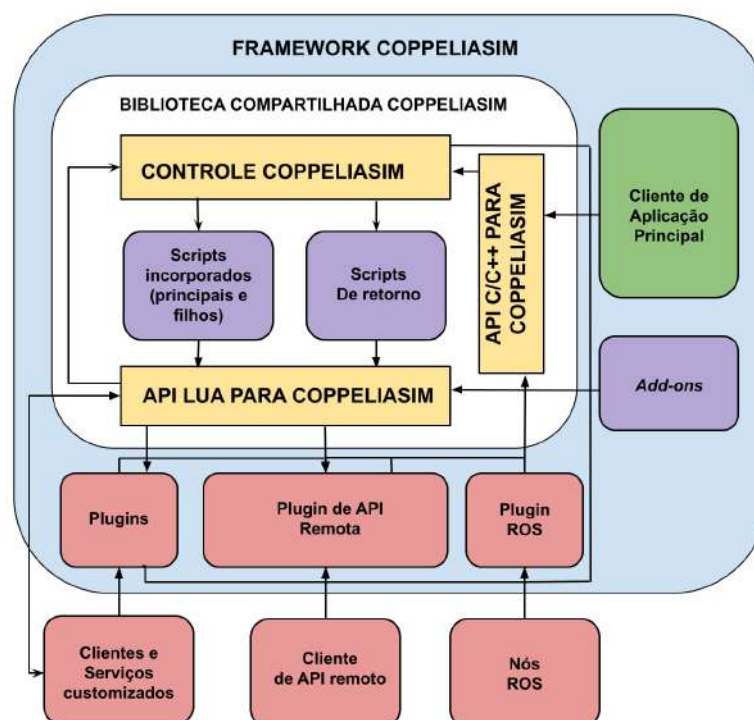


Fonte: Adaptado de Coppelia (2022).

E API de outras funções: funções não nativas do CoppeliaSim, fornecidas por *plugins* específicos. Possuem uma nomenclatura única, com o prefixo *sim* sem o ponto (exemplo: *simOMPL*, *simUI*, *simIK*). Os *scripts* desse grupo são:

- *Plugins*: pacotes escritos para o CoppeliaSim em outra linguagem, sendo usados para fornecer funcionalidades especiais na simulação, implementando importadores ou exportadores específicos, hardwares e interfaces ROS.
- Nós (ROS/ROS2): utiliza de um nó ROS como *plugin* para vínculo e transmissão de dados (via *publishers* e *subscribers*) com os serviços ROS. É considerada a comunicação ideal para aplicações utilizando o ROS, pela flexibilidade e fidelidade na troca de informações, ainda que seja necessário um controle eficaz entre as diferentes mensagens enviadas para o ROS.
- Clientes API remotos: alternativa a interface ROS, utiliza uma API externa (serviços e clientes de API remotos) para chamada de funções remotas. O cliente remoto incorpora em um hardware (de um robô real) um código (de qualquer linguagem) de função para troca de dados com o servidor do CoppeliaSim.

Figura 39: Estrutura de controle V-REP/CoppeliaSim.



Fonte: Adaptado de Rohmer, Singh e Freese (2013).

2.3.3.3 Linguagens de Programação.

- Linguagem de Programação C.

Dennis Ritchie inventou a linguagem de programação C e foi o primeiro a implementá-la utilizando um computador DEC PDP-11, com sistema operacional Unix (SCHILDT, 1997). Conforme Ascencio e Campos (2008) a linguagem C é um processo de evolução de outras linguagens, sendo elas BCPL e B. Com o passar dos anos várias implementações de C foram criadas, com isso muitas discrepâncias foram geradas. Para estabelecer um padrão em 1983 o *American National Standards Institute* – Instituto Nacional de Padrões Americano (ANSI) criou um comitê responsável por padronizar a linguagem C (RITCHIE, 1993).

Um detalhe importante mencionar é que a linguagem C é sensível a letras maiúsculas e minúsculas, portanto, todos os comandos sempre devem ser escritos em minúsculo.

De acordo com Schildt (1997) a linguagem C é uma linguagem de médio nível. O autor interpreta desta maneira, pois, enquanto possui elementos de linguagem de alto nível, também possui funcionalidades do *assembly*.

- Linguagem de Programação C++.

A linguagem de programação C++ foi inicialmente desenvolvida por Bjarne Stroustrup em 1979 nos laboratórios da Bell Labs, para facilitar a organização conjunta de programas da *Simula* com a eficiência da linguagem C (STROUSTRUP, 1998).

Integrando o paradigma de programação *Object-oriented programming* – Programação Orientada a Objetos (OOP) com a linguagem C, Bjarne obteve uma linguagem flexível que fornece abstrações consideradas de alto nível e funcionalidades presentes em linguagem de baixo nível. Suas principais características: ambiente *open source*, curva de aprendizado baixa, controle de recursos na linguagem, garantido rapidez e sobrecarga zero; pode trabalhar sem o sistema operacional, capacidade de adaptação do código ao problema, é adequada para a maioria dos sistemas. (STROUSTRUP, 1998).

Sendo LIBERTY (1999) a estrutura principal do C++ dividida em três pilares principais:

- Polimorfismo: Um operador quanto uma função possibilitando ações diferenciadas sobre o mesmo objeto.

- Herança: “é a capacidade de criar classes que herdam funções e estruturas de dados definidas em outras classes, sendo possível redefinir ou mesmo adicionar novos elementos”.
- Encapsulamento: capturar, em uma mesma classe, objetos que possuam relações de comportamento ou atributos gerais.

Hoje a Linguagem C++ é amplamente utilizada, em aplicações como o desenvolvimento de sistemas embarcados, jogos, sistemas operacionais, sistemas de alto desempenho e outros. Alguns exemplos de *software* desenvolvidos baseados na linguagem C++ : Matlab, Microsoft, editores de imagem (Photoshop), Arduino *Integrated Development Environment* – Ambiente de Desenvolvimento Integrado (IDE) e outros.

- Linguagem de Programação Python.

A linguagem de programação Python foi desenvolvida por Guido Van Rossum em 1982, no *Centrum Wiskunde and Informatica* na Holanda. Guido integrou uma linguagem de *scripting* com a sintaxe da linguagem ABC, criando o Python, capaz de atender a sua necessidade de um melhor tratamento ao administrar o sistema operacional Amoeba na época (FOUNDATION, 2022a).

A linguagem de Guido, é hoje uma das linguagens mais utilizadas e sua popularidade tende a crescer (SRINATH, 2017), citando suas principais características:

- Linguagem de alto nível, legibilidade mais acessível aos programadores, tipagem dinâmica e concisa (MANZANO, 2018);
- Sintaxe suporta vários tipos de paradigmas (orientados a objetos, funcional, imperativo e procedural) (SRINATH, 2017).
- Pode ser integrado a outras linguagens como C, C++ e Java (DAWSON, 2003).
- É gratuito e open source (FOUNDATION, 2022a).

Em relação a sua estrutura Python é usualmente uma linguagem interpretada (SUBASI, 2020). Python pode ser uma linguagem compilada (código fonte é convertido diretamente pelo processador/programa) mas isto depende do tipo de implementação utilizada (DALCÍN; PAZ; STORTI, 2005). Desse modo, na linguagem interpretada, o código fonte do programa é compilado pelo *CPython* (interpretador de *Bytecode* nativo do Python) em *Bytecode* (formato intermediário entre código

fonte, texto manipulado pelo programador, e o código que a máquina executa), este formato que é posteriormente interpretado linha a linha, executando assim cada comando do programa (SUBASI, 2020).

Python é uma ferramenta poderosa e pode ser usada para diferentes aplicações como: desenvolvimento *Web* e Protocolos internet (*Frameworks* como Django e Pyramid / Protocolos de internet, *HyperText Markup Language* – Linguagem de Marcação de Hipertexto (HTML), JSON, *Extensible Markup Language* – Linguagem de Marcação Estendida (XML)), computação científica e numérica (ScipPY, Pandas), Desenvolvimento de Aplicações *Graphical User Interface* – Interface Gráfica do Usuário (GUI)), desenvolvimento de *software* (Scons, Buildbot, Apache), processamento de imagem (OpenCV, Pillow), aplicações CAD e outros (PYTHON SOFTWARE FOUNDATION, 2022).

Neste projeto a linguagem Python é utilizada no sistema operacional do Raspberry, o Raspbian. Segundo a fundação Raspberry Pi, a linguagem Python é especialmente selecionada para este sistema, devido a sua versatilidade de uso (FOUNDATION, 2022b).

- Linguagem de Programação LUA.

A linguagem Lua foi desenvolvida e implementada pelo grupo de Tecnologia em Computação Gráfica da PUC-Rio, no Brasil (LUA, 2022). Projetada para ser embutível (acoplada a programas maiores, e/ou estender a programas escritos em linguagens como Java, C, C++, Fortran, Ada, Perl, Ruby e outras) a linguagem LUA é uma linguagem de *script* com semântica extensível, permite diversos paradigmas (orientada a objetos, funcional, orientada a dados, descrição de dados e procedural)(LUA, 2022). Lua se destaca de outras linguagens de *script* por sua velocidade, desempenho, portabilidade e economia de recursos (IERUSALIMSKY; FIGUEIREDO; CELES, 2007).

As características principais da LUA que a fazem ideal para *scripting* e prototipagem rápida:

- Interpretação dinâmica: Capaz de executar trechos de códigos no mesmo ambiente de execução;
- Tipagem dinâmica forte: capaz de executar verificações de tipos na execução do programa;

- Gerencia automática de memória dinâmica: capaz de auto gerenciar memória no programa, sem necessidade de liberar memória. (LUA, 2022) (IERUSALIMSKY, 2022).

É a linguagem de *script* mais utilizada no desenvolvimento de jogos (por empresas como LucasArts, BioWare, Microsoft, Monkeystone Games), além de ser utilizada também em sistemas embutidos e *software* s gráficos (CELES; FIGUEIREDO; IERUSALIMSKY, 2004).

A linguagem Lua é também utilizada para programar os *scripts* no simulador Coppeliasim, este utilizado neste projeto.

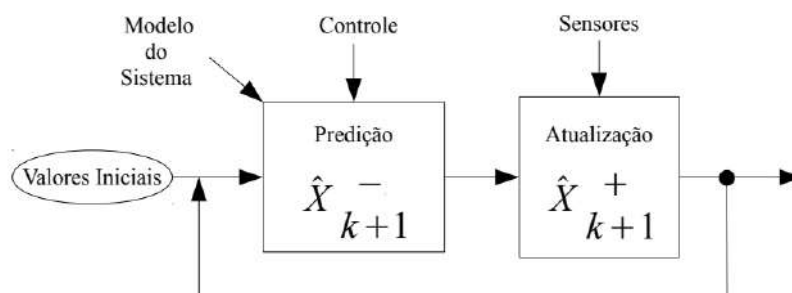
2.3.3.4 Filtros Digitais

No processamento digital de sinais, filtros realizam operações matemáticas conforme o tempo amostrado. Dessa forma, é possível trabalhar com os sinais mais estáveis no sistema.

- Filtro Kalman.

De acordo com Bongard (2008), o Filtro de Kalman é uma das ferramentas de método repetitivo que estima o estado de um sistema dinâmico na presença de ruído. Consegue produzir estimativas dos valores reais das medições e os seus valores calculados associados ao prever um valor, estimar a incerteza do valor previsto, bem como o cálculo de uma média ponderada do valor esperado e o valor aferido. O maior peso é dado para o valor com o menor grau de incerteza.

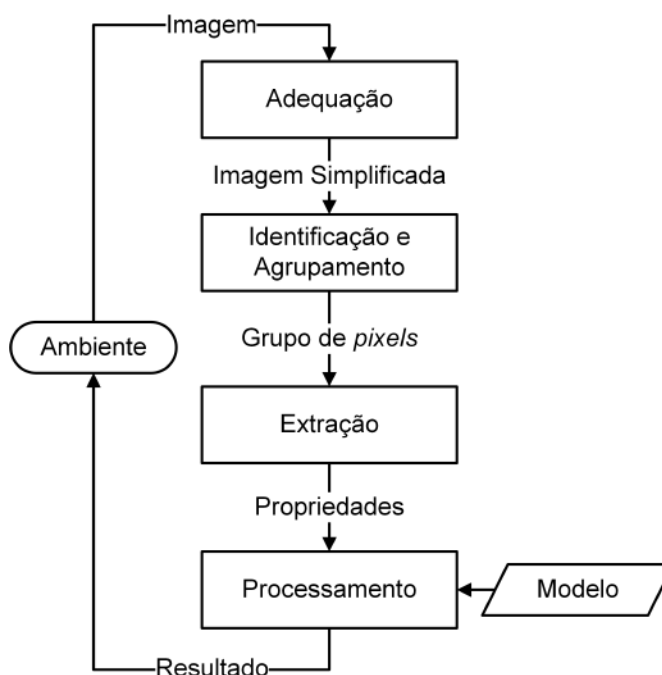
A estimativa produzida pelo método tende a estar mais próxima da realidade do que as medições iniciais, dado que a média ponderada tem uma incerteza de estimativa melhor do que qualquer um dos valores individuais que realizam essa mesma média. A Figura 40, demonstra como ocorre a modelagem do sistema de Kalman em dois estágios (BONGARD, 2008).

Figura 40: Estrutura do Filtro de Kalman.

Fonte: Adaptado de (BONGARD, 2008).

- Filtro Digital de Imagens.

Siegwart, Nourbakhsh e Scaramuzza (2004) diz que a interpretação visual é um problema extremamente desafiador para se resolver completamente. Pesquisas evoluíram significativamente ao longo das últimas décadas, para inventar algoritmos que entendam uma cena com base em imagens 2D. Cobrir o campo da visão computacional e do processamento de imagens está, obviamente, além do escopo deste trabalho. A Figura 41, tenta demonstrar um fluxo de como ocorre o processamento digital de imagens.

Figura 41: Esquema do processamento de imagem computacional.

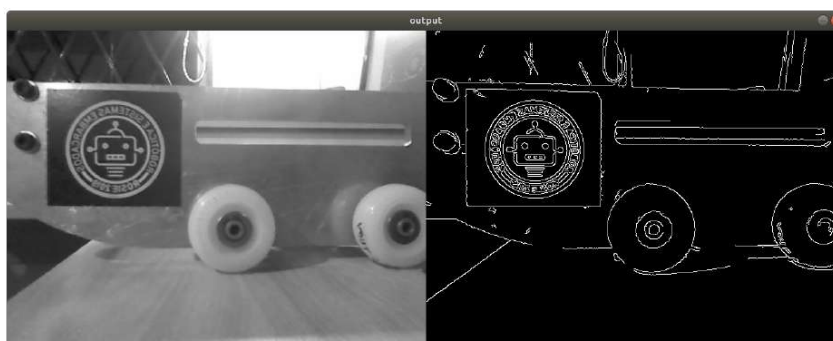
Fonte: Adaptado de Siegwart, Nourbakhsh e Scaramuzza (2004).

Nesse sentido, têm-se algumas aplicações estabelecidas, como o modelo de detecção de arestas Canny, que tratou a detecção de borda como um processamento de sinal no qual existem três objetivos explícitos:

- Maximização da relação sinal/ruído;
- Alcançar a maior precisão possível na localização das arestas;
- Minimizar o número de respostas de borda associadas a cada borda.

Na prática, o extrator de arestas Canny suaviza a imagem e a diferenciação são combinados em uma operação matemática, conforme é possível notar na aplicação do filtro na Figura 42 (XU; BAOJIE; GUOXIN, 2017).

Figura 42: Exemplo do algoritmo de Canny aplicado.



Fonte: Autor (2022).

2.4 GÊMEO DIGITAL

Um DT é uma representação de um ativo real, ou seja, uma réplica virtual de um objeto físico ou um processo, conforme diz Rosen (2015). Já Singh (2021) fala que pode ser uma parte, um subsistema com diversos componentes ou um sistema feito de subsistemas, todos virtualizados, simulando e representando sua condição atual, com transferência bidirecional de dados (quantitativos, qualitativos, dados históricos, dados ambientais e dados em tempo real) entre a contraparte física e digital.

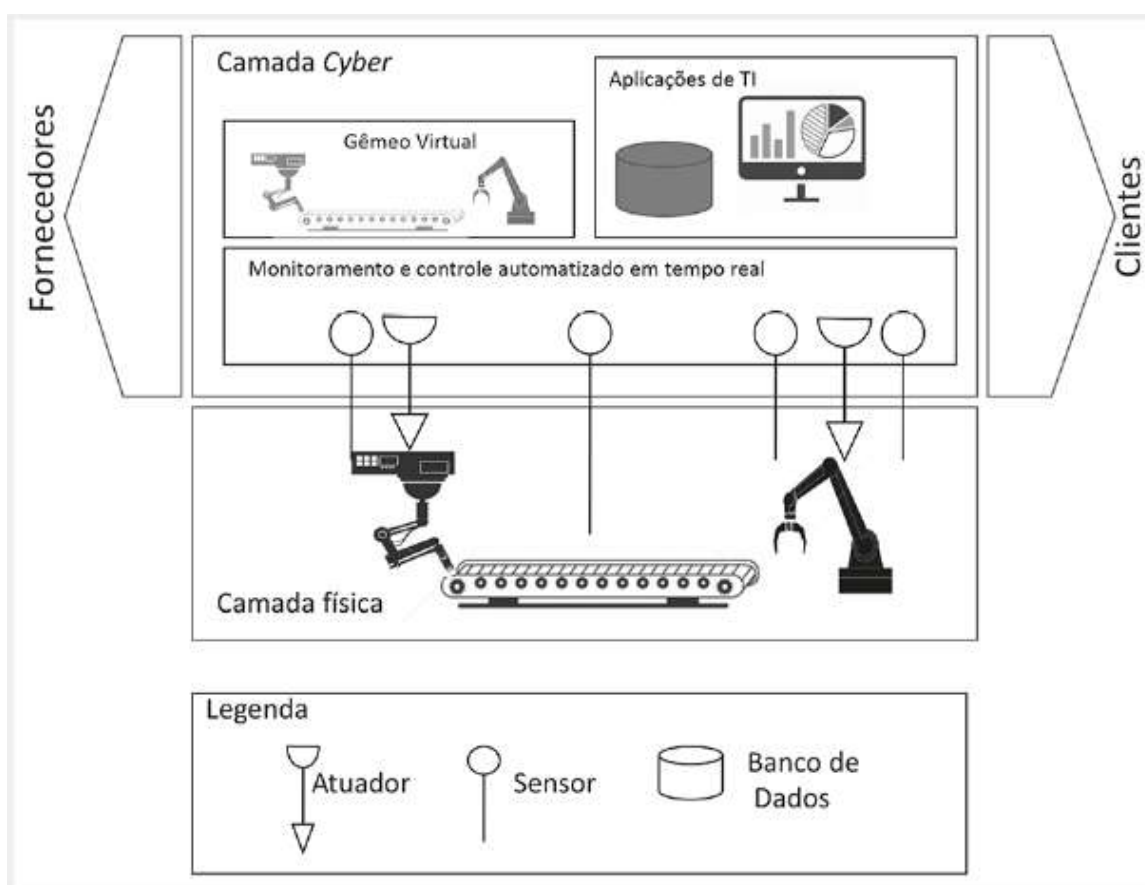
Com essa virtualização, é possível simular as mais diversas situações de operação. Os ativos virtuais podem ser modificados e otimizados com muito mais facilidade, menor custo e maior velocidade, até se encontrar o resultado mais adequado para o fim que se busca (ARMENDIA, 2019).

Segundo Armendia (2019), os DTs podem ser utilizados nas mais diversas aplicações, desde o desenvolvimento de produtos até a melhoria de processos ou

de logística. Por se tratar de modelos virtuais, podem conjugar com as novas tecnologias da indústria 4.0, como os CPS, IoT e CC.

A arquitetura de controle dos DTs (Figura 43) é dividida em duas camadas principais: a camada física (formado pelos sistemas físicos, objetos a serem monitorados) e a camada ciber (sistemas virtuais, cópias digitais dos objetos, sistemas para coleta, troca, monitoramento e controle dos dados) (SACOMANO, 2018).

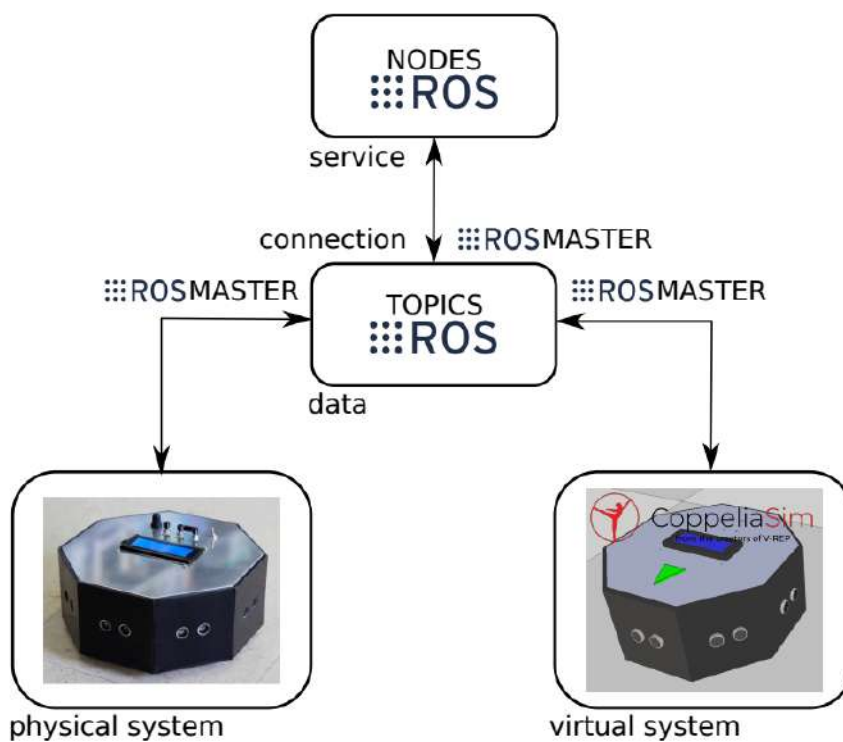
Figura 43: Arquitetura CPS com DT.



Fonte: Adaptado de (SACOMANO, 2018).

Aplicadamente, tem-se o artigo de Magrin, Del Conte e Todt (2021) que, através do desenvolvimento de um robô móvel, implementou o *framework* ROS, como interface de integração para a criação de um DT, conforme mostrado na Figura 44.

Figura 44: Proposta de um gêmeo digital utilizando o ROS

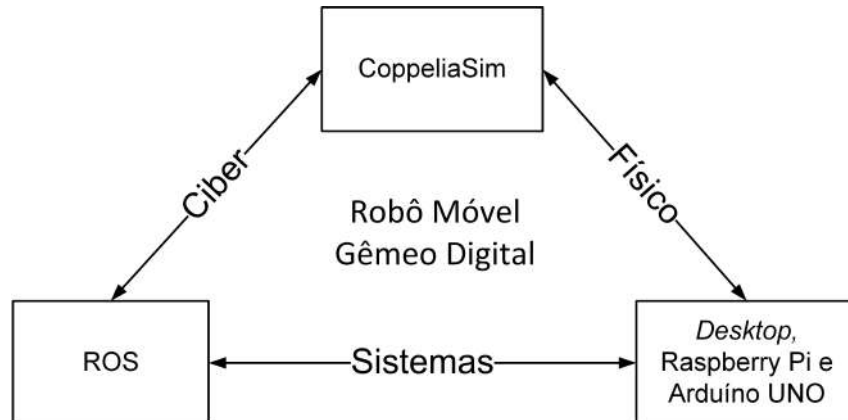


Fonte: Adaptado de Magrin, Del Conte e Todt (2021).

3 DESENVOLVIMENTO

O desenvolvimento deste projeto, visa, com base na tríade apresentada por Sacomano (2018), a integração dos CPS com o intuito de transformá-lo em um DTMR proposto anteriormente por Magrin, Del Conte e Todt (2021) (Figura 45).

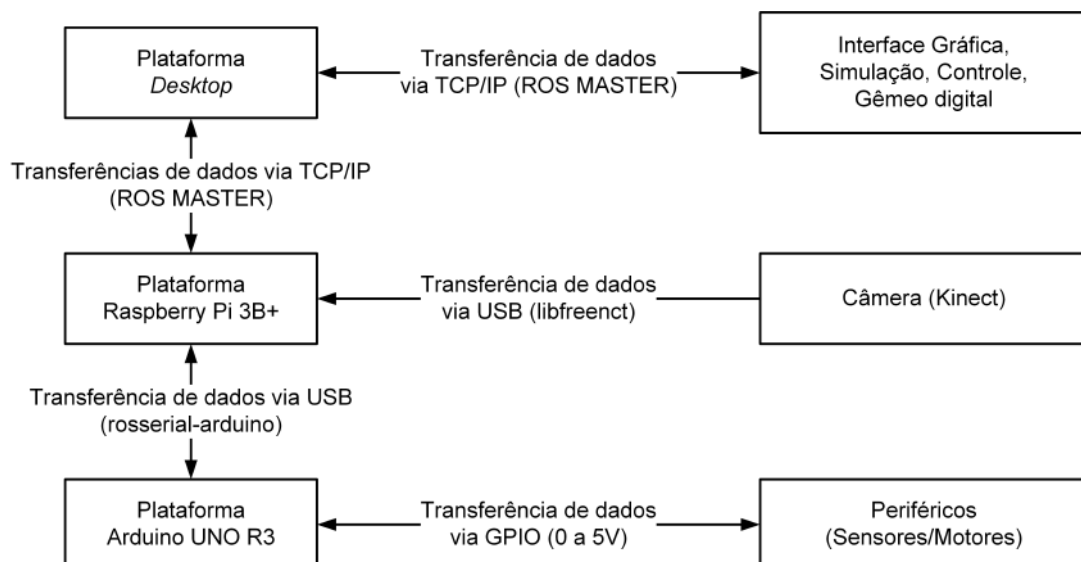
Figura 45: Tríade DTMR Tupy



Fonte: O Autor (2022).

Especificadamente, através da Figura 46, o fluxograma demonstra a integração ciber-física destas interfaces.

Figura 46: Fluxograma geral de integração DTMR Tupy.



Fonte: O Autor (2022).

3.1 SISTEMA FÍSICO

Nesta seção, será mostrado todo o desenvolvimento do sistema físico do DTMR Tupy, constituído das seguintes atividades: adequação, construção, modelagem, e montagem do modelo virtual do robô; implementação, integração dos periféricos no sistema físico do robô, conforme planejamento da Figura 46.

3.1.1 Implementação de melhorias na plataforma pré-desenvolvida

Este trabalho consiste no desenvolvimento contínuo do projeto robô Tupy do departamento ROSIE da Unicuritiba. Desta forma, houve a necessidade de fabricar alguns componentes para adaptar as plataformas RPi 3b+ e o Arduino UNO. Anteriormente, o robô se encontrava com o *hardware* do *Peripheral Interface Controller* – Controlador de Interface Periférica (PIC) e sem os suportes (Figura 47). O uso destas plataformas, se deu, principalmente, pela otimização da questão de integração dos componentes com serviços virtuais, por bibliotecas pré-compiladas.

Figura 47: robô Tupy versão 1

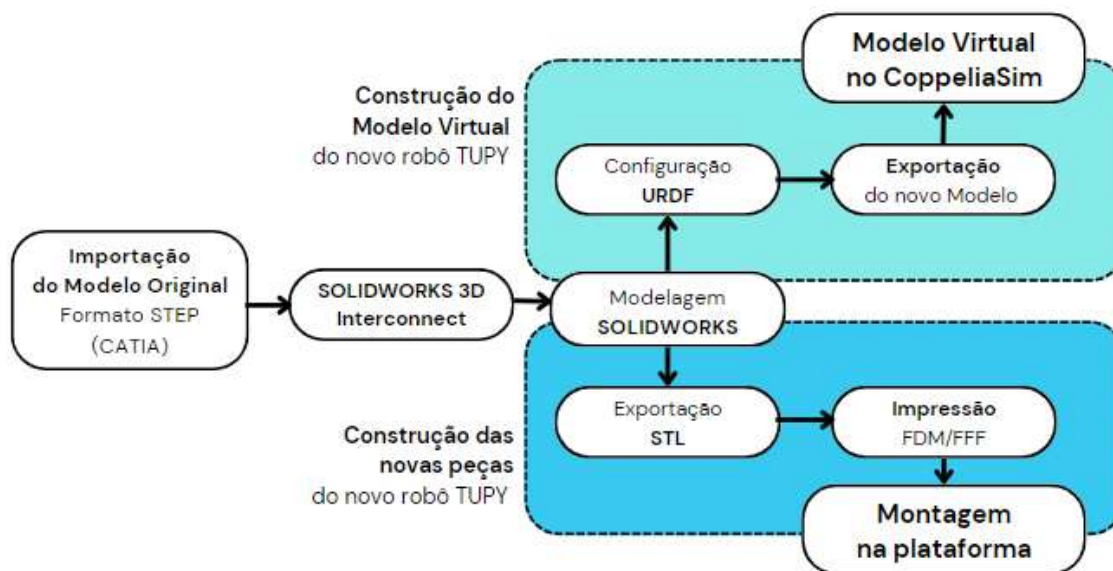


Fonte: O Autor (2022).

As modificações desenvolvidas na plataforma robótica do robô Tupy visam adicionar melhorias e adequar a plataforma para este projeto. Desta forma, o andamento desse processo é dividido em duas partes principais a partir do modelo 3D

no software SolidWorks: a construção das novas peças do DTMR Tupy, e a construção do modelo virtual do DTMR Tupy, conforme a Figura 48. Os detalhes dessa estrutura serão descritas com detalhes nos tópicos 3.1.1, 3.1.1 e 3.1.3.1 .

Figura 48: Estrutura do desenvolvimento para o sistema mecânico do DTMR Tupy



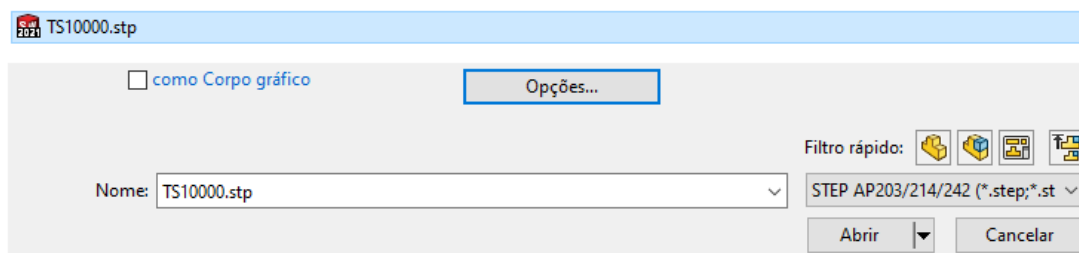
Fonte: O Autor (2022).

- Importação e adequação do modelo original.

Como esse projeto utiliza o software SolidWorks versão 2021, a nova modelagem do robô Tupy inicia-se da importação do arquivo original de montagem que foi elaborado no software CATIA versão *V5 Student Edition*, segundo Magrin, Del Conte e Todt (2021).

O arquivo exportado do CATIA tem o formato padrão em *Standard for the Exchange of Product Data* – Norma para Troca de Dados de Produtos (STEP), de norma ISO 10303-2xx e precisa ser convertido para a utilização. O próprio SolidWorks possui uma ferramenta chamada *SOLIDWORKS 3D Interconnect* que permite a leitura de diferentes extensões. Dessa forma, ao adicionar o arquivo, define-se o como o modelo **STEP AP203/214/252**, mostrado na Figura 49.

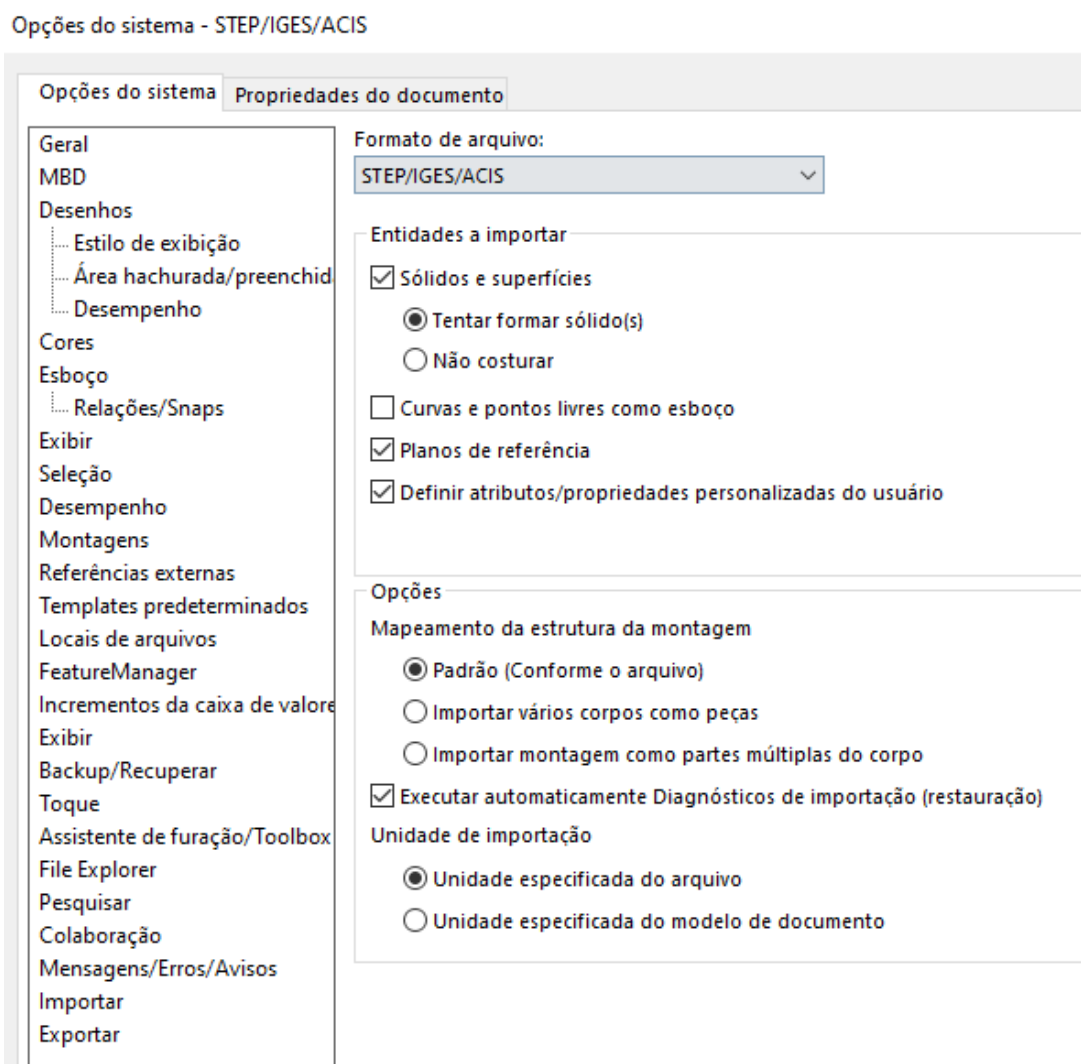
Figura 49: Seleção de tipo de arquivo STEP para o SolidWorks 2021.



Fonte: O autor (2022).

Na sequência, faz-se uma seleção de algumas propriedades disponíveis em relação à importação escolhida. Como mostra a Figura 50, os campos foram preenchidos visando o padrão fornecido pelo próprio desenvolvedor em seu fórum.

Figura 50: Seleção de propriedades de importação STEP no SolidWorks 2021.



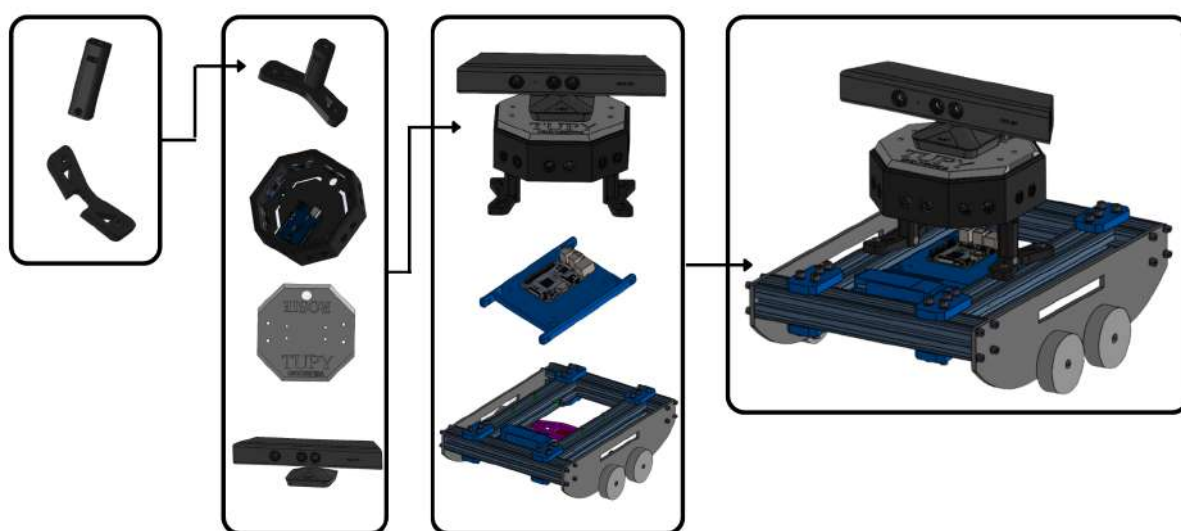
Fonte: O autor (2022).

Um ponto importante a acrescentar, é que esse tipo de importação não permite a alteração do esboço original dos componentes. Nesse sentido, com a modelagem STEP aberta no SolidWorks pode-se apenas suprimir ou excluir as peças que não serão mais utilizadas.

- Desenvolvimento de novas peças

Para o desenvolvimento das novas peças, serão apresentadas conforme a sequência vista na estrutura da Figura 51. Sendo assim, a modelagem final do DTMR Tupy será composta por: a construção do novo bloco que sustenta os sonares e o Kinect, e a combinação desse bloco com o novo suporte para o RPi e da antiga base da primeira versão do robô Tupy.

Figura 51: Estrutura para a construção do modelo 3D no SolidWorks do DTMR Tupy.

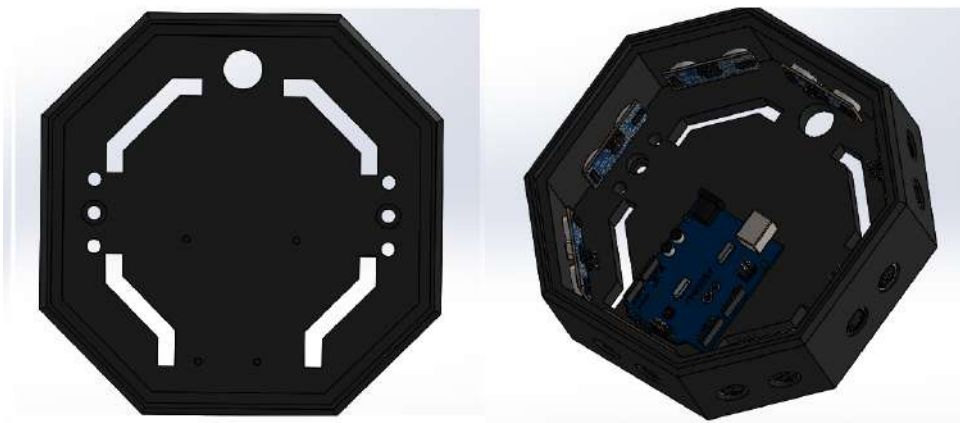


Fonte: O autor (2022).

O conjunto que formava o suporte dos sonares e do *Kinect* era anteriormente uma peça única, exceto a parte superior que fechava o octógono que era independente. Dessa forma, repensou-se a estrutura, de modo a obter melhor flexibilização, separada em três partes: suporte para sonares octogonal, suporte de sustentação e base.

Para o octógono, foram realizadas revisões nas medidas para comportar corretamente os sensores ultrassônico HC-SR04, e inseridos ressalto de (3 mm) de comprimento para guiar o encaixe da plataforma Arduino UNO R3. Além disso, adicionou-se duas furações com rebaixo (M6) cada para fixação com os suportes de sustentação, conforme mostra a Figura 52.

Figura 52: CAD — Suporte para sonares.



Fonte: O autor (2022).

Para o suporte de sustentação, o comprimento total da peça foi alterado para (65 mm), a fim de obter uma altura maior em relação ao chão e evitar interferências na leitura do Kinect (com as peças separadas há a possibilidade de refazer as medidas caso necessário, trocando a peça por outra maior ou menor). No topo do suporte foi inserido um furo (M6) e um corte lateral de (10 × 6,5 mm) para encaixe da porca (M6) que fixará o parafuso.

Para a base, foram adicionados uma furação (M6) na região central pela lateral da peça, para esta ser fixa com o suporte de sustentação, mostrado na Figura 53.

Figura 53: CAD 3D — suporte e base para octógono.



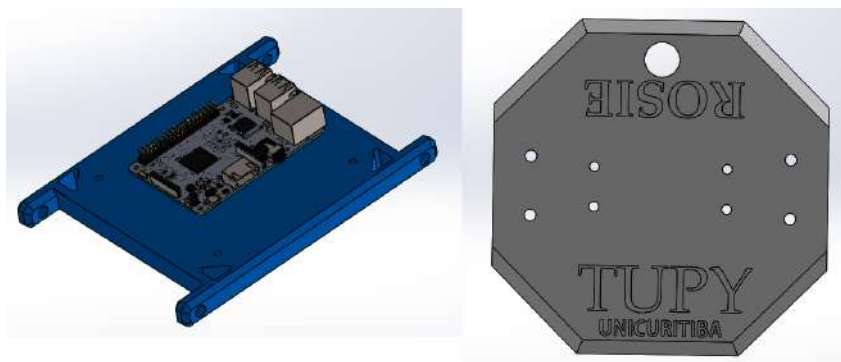
Fonte: O autor (2022).

Além disso, para acoplar a plataforma RPi de maneira adequada, foi modificado a placa que anteriormente suportava o PIC no projeto inicial, ficando a configuração final conforme a Figura 54 à esquerda.

A fim de garantir uma boa estabilidade para o Kinect, a base do topo também foi modificada, como mostra a Figura 54 à direita. Seu conceito foi refeito, assemelhando-se a uma “tampa” para encaixar no octógono, garantindo uma maior

estabilidade. No centro da tampa foram inseridos 4 furos (M3) para fixação da base do Kinect. Finalizado com acabamento por escritas referentes ao nome do robô (TUPY), ROSIE e Unicuritiba.

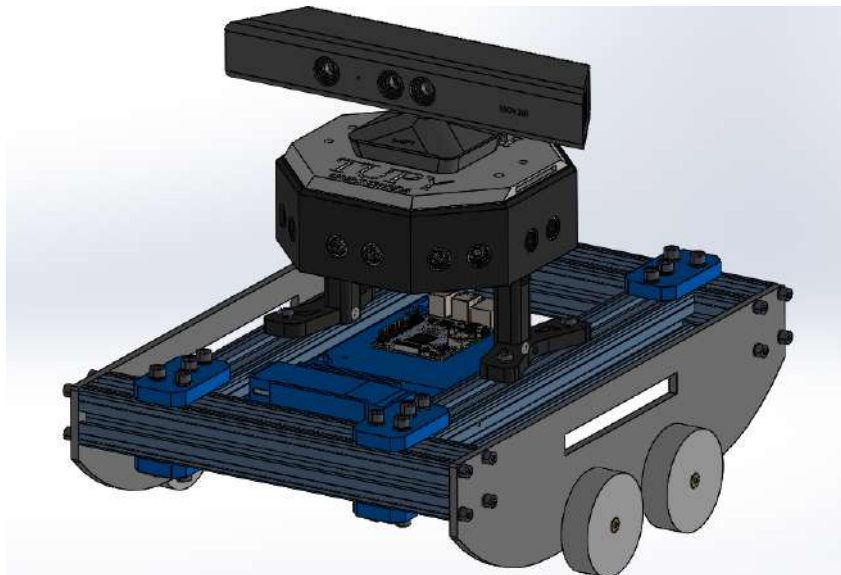
Figura 54: CAD 3D — Suporte RPi (esquerda) / Tampa para octógono (direita).



Fonte: O autor (2022).

Por fim, o desenho mecânico 3D completo do robô Tupy montado com as peças modificadas ficará como mostra a Figura 55.

Figura 55: CAD 3D — montagem final do robô Tupy.



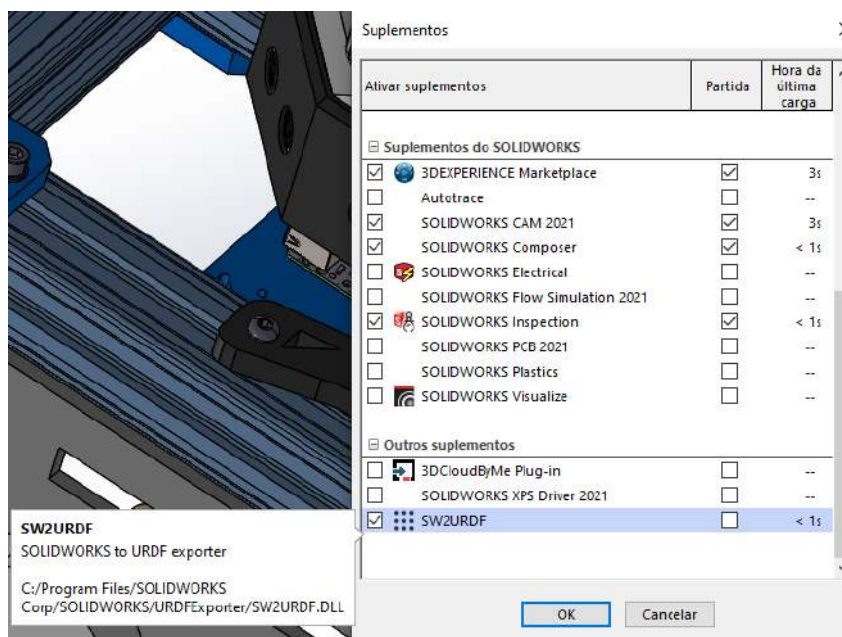
Fonte: O autor (2022).

- Exportação do novo modelo para o CoppeliaSim.

Para visualizar o novo modelo do robô Tupy no simulador CoppeliaSim, é necessário exportar o arquivo em formato *Unified Robot Description Format* – Formato de Descrição do Robô Unificado (URDF). O próprio ROS tem um sistema específico para exportar peças do SolidWorks nesse formato chamado *SW2URDF*. Esse

é um *add-in* que permite a exportação de peças e conjuntos, contendo um diretório para texturas, configurações, *meshes*, e arquivos para robôs. Assim, o exportador é instalado no SolidWorks como um recurso, ativado na aba ferramentas. Conforme mostra a Figura 56.

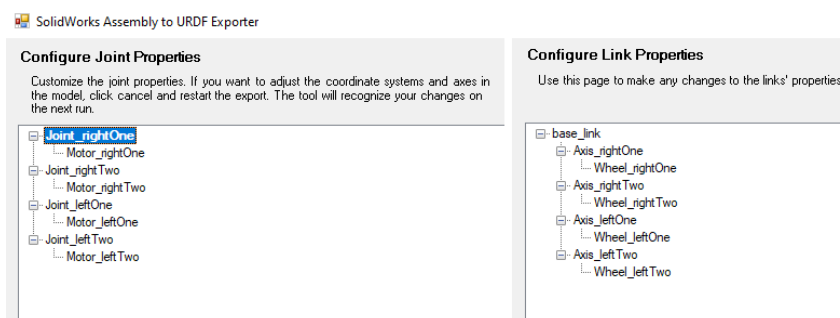
Figura 56: Ferramenta de exportação SolidWorks para URDF.



Fonte: O autor (2022).

Na sequência, é necessário configurar as ligações para as partes móveis do robô Tupy (eixos e rodas) na página de gestão de propriedades do exportador URDF. Cada ligação terá um nome único, que será composto de um conjunto único, de componentes/peças, formando a arborescência (que pode chegar até no máximo de dois níveis de profundidade). Ligações são as partes rígidas do robô, ligadas umas, as outras por juntas. Essas juntas se movem permitindo que haja um movimento entre as ligações ligadas. Para as rodas do Tupy temos articulações do tipo revolutivas (movimento rotativo), enquanto seu eixo se torna uma ligação. Conforme mostra Figura 57 a configuração final de propriedades das ligações do robô.

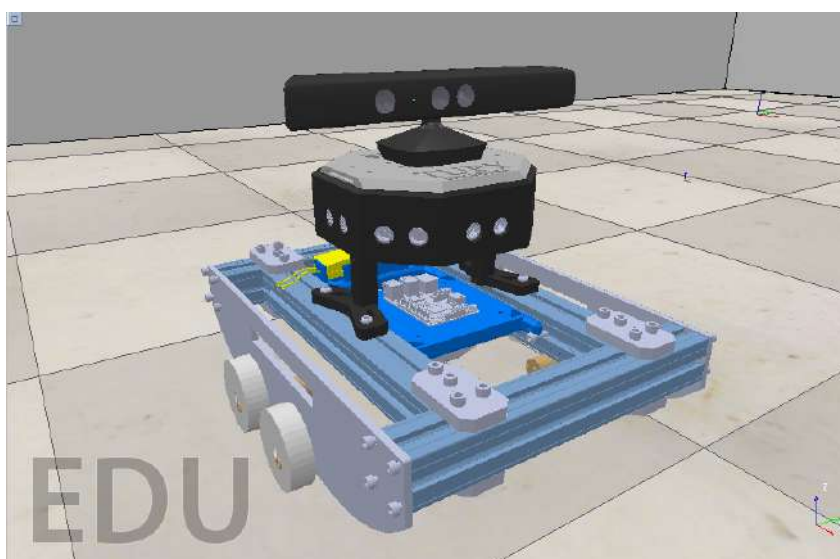
Figura 57: Configuração de juntas e ligações do robô no exportador URDF.



Fonte: O autor (2022).

Assim, as peças da montagem serão exportadas separadamente em URDF (esse processo é necessário, pois assim obtém-se maior equivalência das texturas originais criadas no SolidWorks) e construídas novamente no simulador CoppeliaSim. Conforme mostra a Figura 58.

Figura 58: Modelo virtual DMTR Tupy no CoppeliaSim.



Fonte: O autor (2022).

3.1.2 Integração Física

A integração física do robô Tupy se deu por mudanças e melhorias elencadas, principalmente, em:

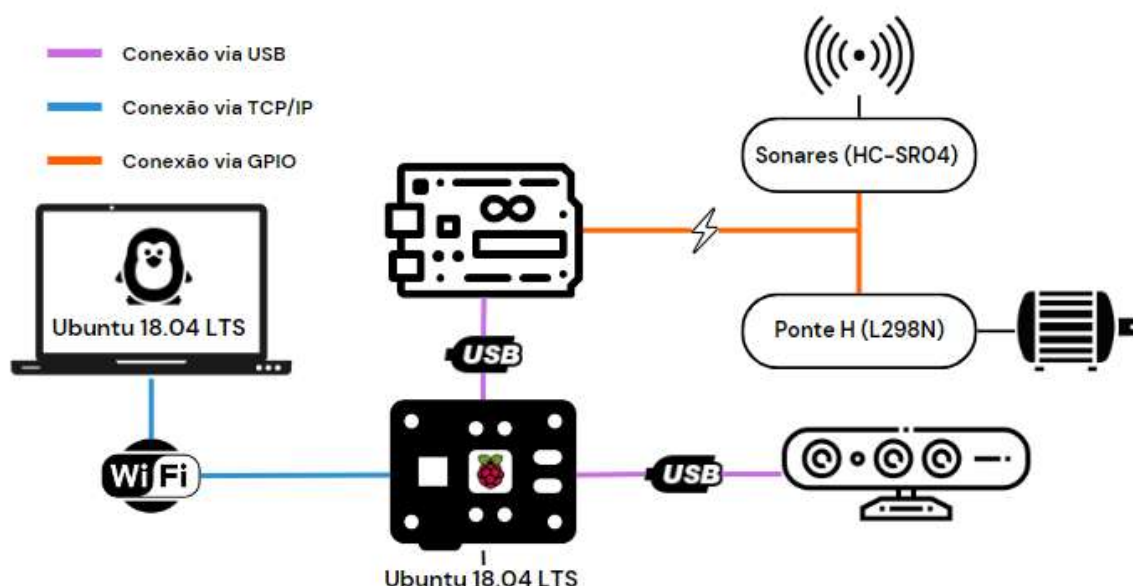
- Substituição do microcontrolador PIC (18F4550) pela plataforma Arduino UNO R3 com (ATmega328P), devido à implementação otimizada com o ROS, através do pacote (rosserial-arduino). Além disso, a plataforma oferece facilidade

na gravação do *firmware*, através do *bootloader* integrado com a IDE *open source*;

- Adição do microcomputador RPi 3B+ para suprir à demanda de troca de dados em alto nível e controle remoto, através do protocolo TCP/IP. Leitura e escrita dos periféricos de baixo nível, através da comunicação serial (USB) com o ATmega328P;

Assim as alterações no sistema eletrônico podem ser vistas conforme a estrutura de desenvolvimento presente na Figura 59.

Figura 59: Estrutura do sistema eletrônico do DTMR Tupy.

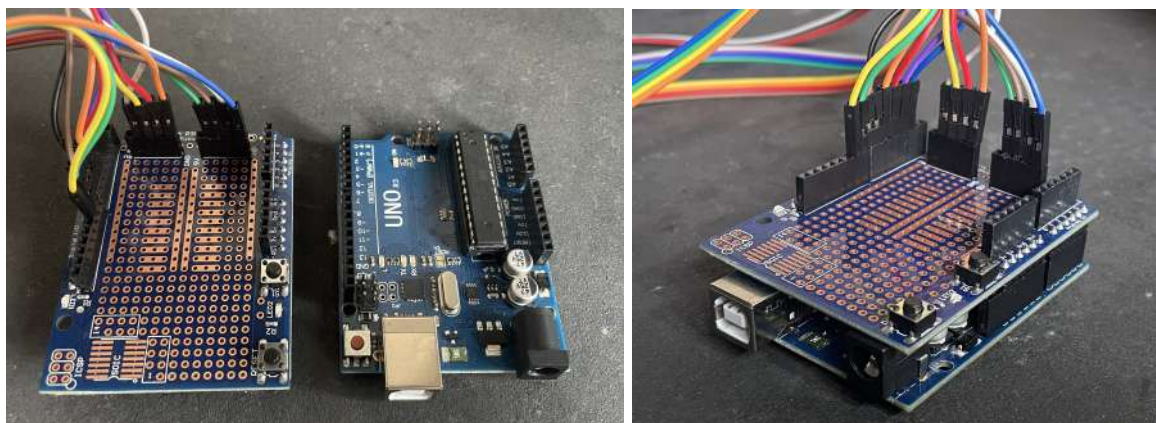


Fonte: O autor (2022).

3.1.2.1 Integração: Arduino e Protoshield V5

A fim de melhorar a versatilidade da plataforma Arduino, tornando mais seguro contra interferências mecânicas externas e organização das conexões elétricas, utilizou-se o módulo eletrônico Protoshield V5 (Figura 60). Esta placa oferece uma área de prototipagem para realizar pequenas montagens e conexões pelos terminais fêmeas, principalmente.

Figura 60: Arduino e Protoshield.

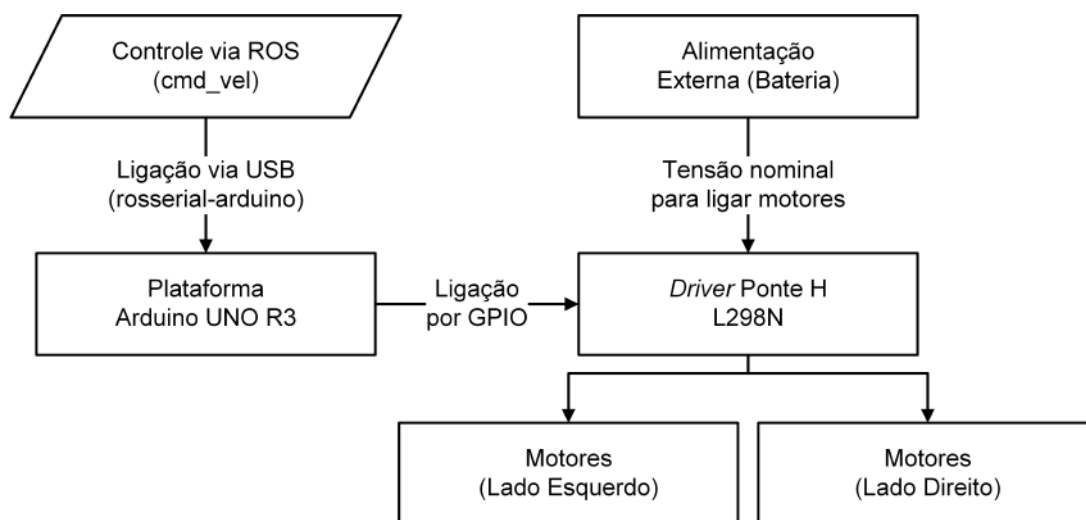


Fonte: O autor (2022).

3.1.2.2 Integração: Arduino e Ponte H (L298N) com Motores Elétricos

O controle dos motores, utilizou-se de um módulo comercialmente conhecido como *driver* de Motor Ponte H L298N, simplificando, assim, a eletrônica de potência do projeto, já que a plataforma Arduino, tampouco seu microcontrolador, não são suficientes para suprir a demanda da potência para os motores. O fluxograma demonstra a lógica de ligação, conforme Figura 61.

Figura 61: Fluxograma de integração: Arduino — Ponte H L298N — Motores.



Fonte: O Autor (2022).

Para a ligação física do componente, utiliza uma bateria conectada nos pinos VCC e GND do *driver* L298N. É importante, também, garantir o nivelamento lógico ao acionar os motores, realizando a equipotencialidade dos terminais de referência GNDs dos periféricos. Já a interface de controle, deve ser feita a partir da conexão

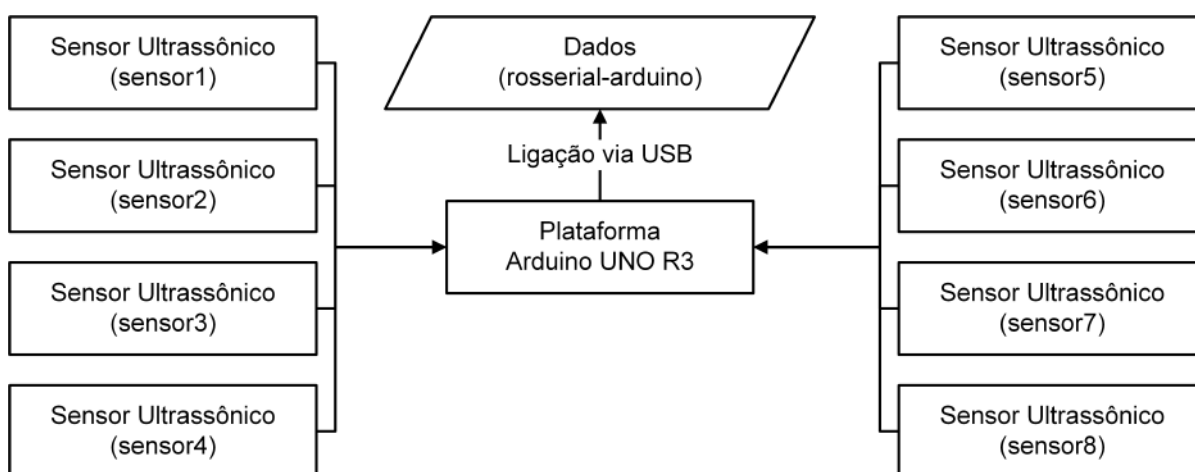
dos pinos (10, 11, 12 e 13) do microcontrolador com o *driver* pelos terminais (IN1, IN2, IN3 e IN4) (Apêndice A), conforme definido na IDE de programação do Arduino, vide Apêndice D.

Observação: os pinos (ENA) e (ENB) poderiam ser controlados via *Pulse Width Modulation* – Modulação por Largura de Pulso (PWM) para um controle mais refinado da velocidade dos motores. Porém, não foi optado por esse controle, já que a movimentação do robô foi feita com o chaveamento direto do motor com 5V, não apresentando maiores problemas. Sendo assim, foi necessário apenas a adição de ‘*jumpers*’ na ligação destes pinos (ENA/ENB). Uma questão positiva, foi a preservação de duas saídas do microcontrolador ATmega328p, já que não foram utilizadas.

3.1.2.3 Integração: Arduino com Sonares (HC-SR04)

Para o sensoriamento do robô, utilizou-se apenas quatro sensores ultrassônicos de modelo HC-SR04, demonstrativamente, com foco na visualização da leitura destes sensores via ROS. As ligações dos sonares foram feitas diretamente no ATmega328P, desprezando qualquer lógica *software/hardware* que pudesse realizar a multiplexação¹ dos sinais para otimização das entradas e saídas do microcontrolador. A Figura 62 exemplificam, teoricamente, como ficaria a lógica de ligação destes sensores. Para mais detalhes, verificar Apêndice A.

Figura 62: Fluxograma de integração Arduino — Sonares (HC-SR04).



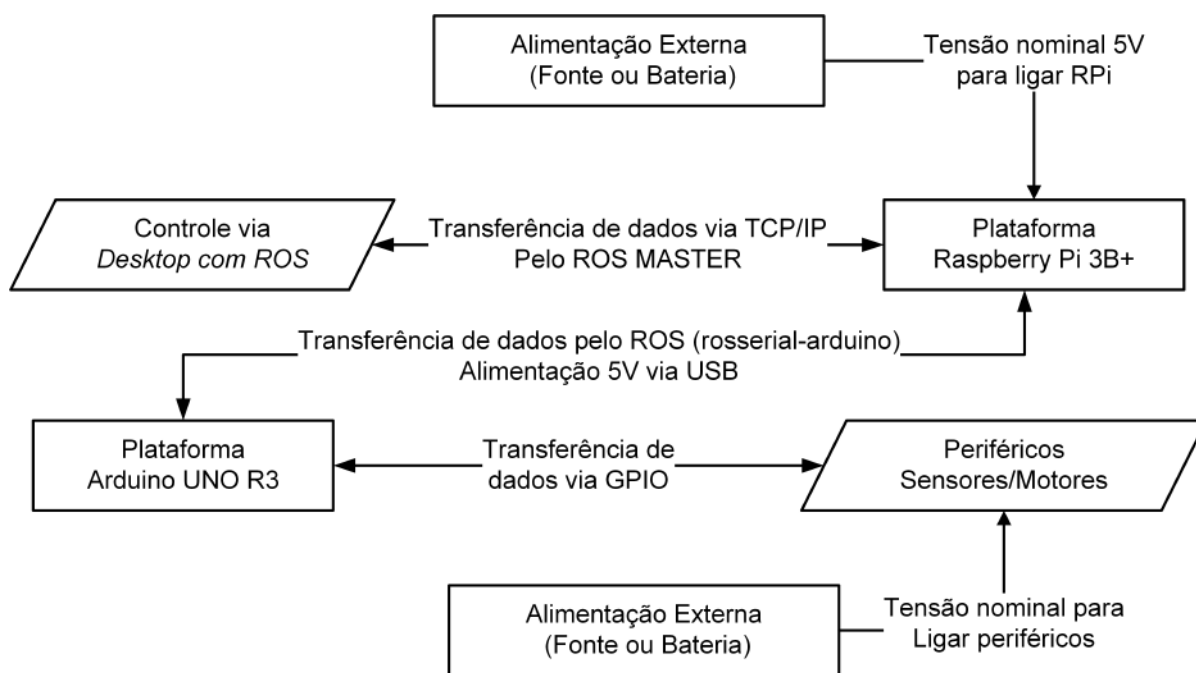
Fonte: O Autor (2022).

¹Nota de texto: a multiplexação é uma técnica que consiste na combinação de dois ou mais canais de informação por apenas um meio de transmissão.

3.1.2.4 Integração: Arduino com RPi 3B+

Para realizar a integração física entre as plataformas Arduino UNO R3 e RPi 3b+, apropriou-se do uso da comunicação serial feita pelas portas USBs dos dispositivos, simplificado a ligação. Uma das vantagens de utilizar a ligação USB é que o próprio RPi realiza a alimentação 5V do Arduino. O fluxograma (Figura 63), demonstra a lógica de ligação.

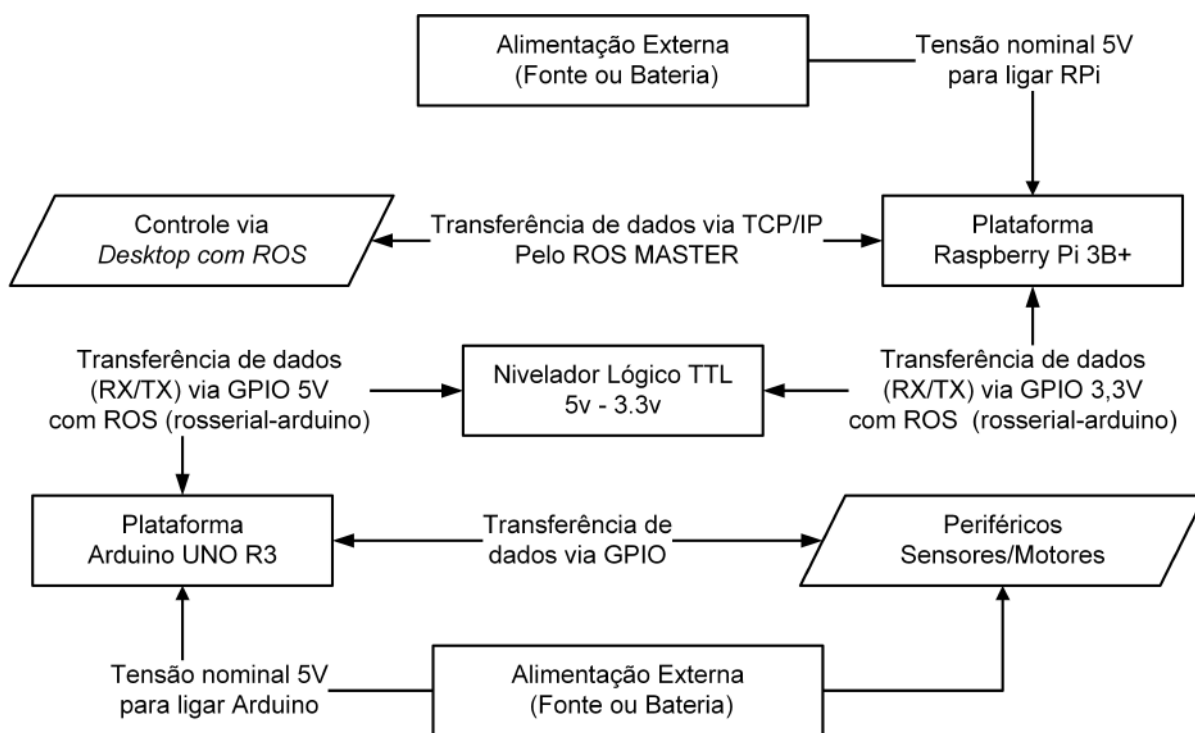
Figura 63: Fluxograma de integração: *Desktop*, Arduino e RPi 3B+ via USB.



Fonte: Autor (2022).

Outra forma possível de realizar a comunicação serial é por conexão via *General Purpose Input/Output* – Entrada/Saída de Uso Geral (GPIO), utilizando os pinos RX/TX de ambos microcontroladores e cruzando os cabos. Porém, se faz necessário a utilização de um nivelador lógico *Transistor-Transistor logic* – Lógica Transistor-Transistor (TTL) e alimentação externa, já que cada plataforma opera em uma tensão diferente, conforme o fluxograma mostrado na Figura 64.

Figura 64: Fluxograma de integração: *desktop*, Arduino e RPi 3B+ via GPIO.

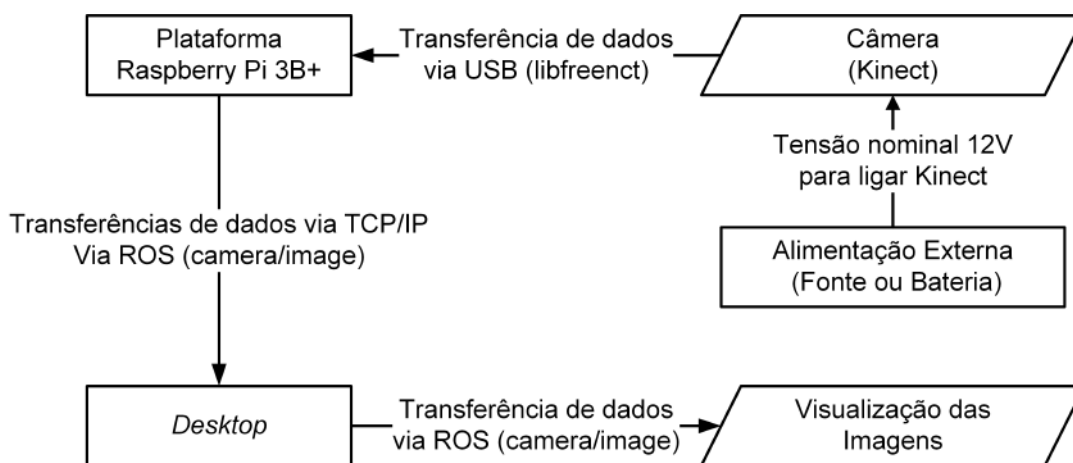


Fonte: Autor (2022).

3.1.2.5 Integração: Kinect com RPi 3B+

Assim como a ligação USB do Arduino com o RPi, também utilizou-se da comunicação USB para integração do sensor Microsoft Kinect Xbox 360 — Versão 1. Já a alimentação do Kinect deve ser feita em 12V, demonstrado no fluxograma de integração (Figura 65).

Figura 65: Fluxograma de integração: Kinect — RPi.



Fonte: O Autor (2022).

3.1.3 Montagem eletromecânica do robô Tupy

Após o desenvolvimento '*offline*' de toda integração física, passou-se para a etapa de montagem eletromecânica dos componentes do projeto, constituídos em:

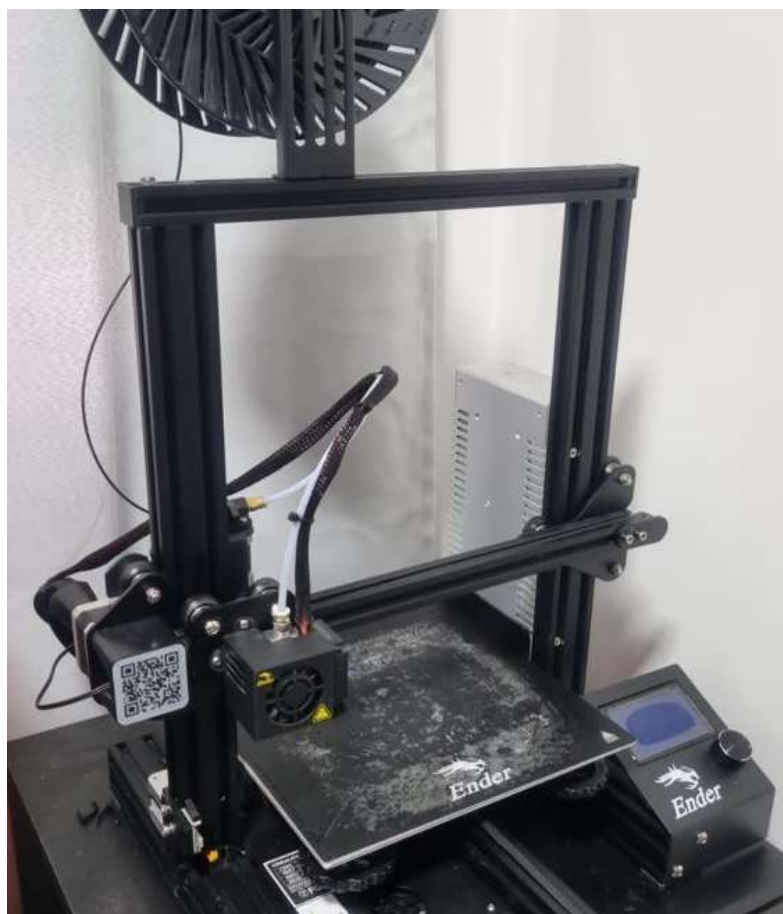
3.1.3.1 Mecânica

A etapa mecânica se constitui na fabricação, ajustagem e montagem dos componentes.

- Impressão 3D das peças mecânicas.

Para confeccionar as peças deste projeto, utilizou-se da tecnologia manufatura aditiva. As peças foram desenvolvidas via *software* SolidWorks para modelagem 3D, e posteriormente exportadas em formato STL para impressora 3D (Figura 66). O processo de manufatura aditiva utilizado para impressão é o método (FDM/FFF). O material utilizado foi o PLA.

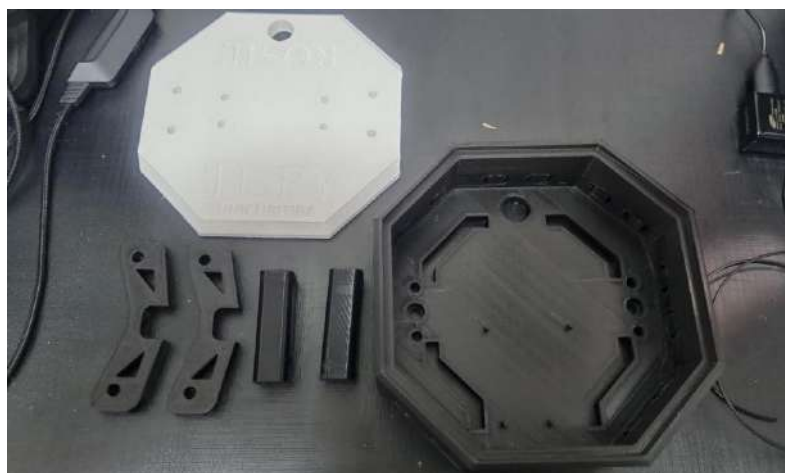
Figura 66: Impressora 3D Creatily Ender 3.



Fonte: O Autor (2022).

O resultado da impressão pode ser visto na Figura 67. Assim, as peças novas podem ser acopladas e substituídas quando necessário. A grande vantagem, é simplificar a complexidade processo de fabricação mecânica, reduzindo o material e o tempo de fabricação, com a separação das peças.

Figura 67: Peças finalizadas.



Fonte: O Autor (2022).

- Ajustagem mecânica da impressão 3D.

Após a impressão, o material fica com bastante rebarbas nas arestas. Por isso a ajustagem mecânica é fundamental no êxito da montagem. A Figura 68, mostra essa condição.

Figura 68: Acabamento da furação dos sensores após impressão.



Fonte: O Autor (2022).

Dessa forma, foi necessário as arestas e, com uma micro retifica, realizar o ajuste da furação para o encaixe dos sensores ultrassônicos, conforme mostrado pela Figura 69.

Figura 69: Furação ajustada.

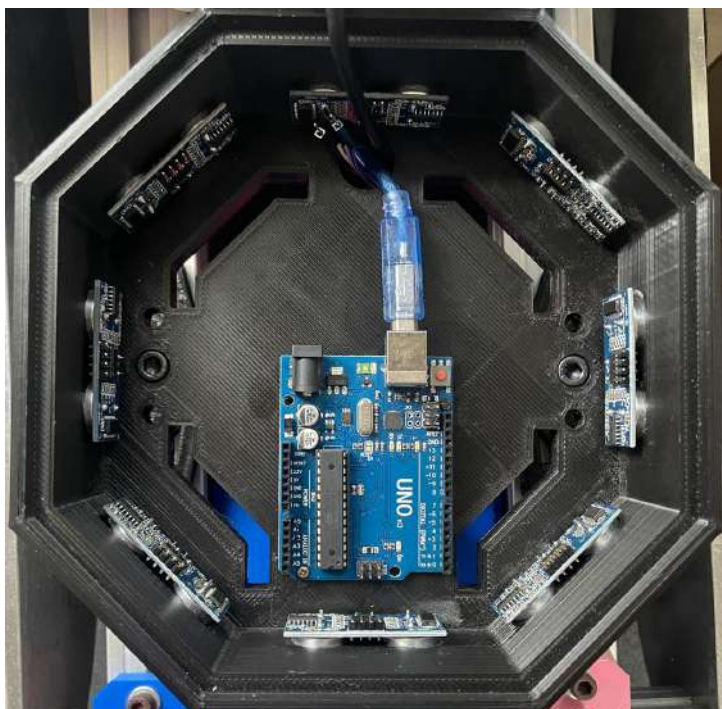


Fonte: O Autor (2022).

- Montagem do suporte octogonal e fixação dos componentes.

Após os ajustes mecânicos realizados, passa-se a montar e fixar os componentes eletrônicos internamente. Devido à precisão da impressora, não houve a necessidade de fixar os sensores com parafusos, pois o encaixe ficou perfeito. Já o Arduino necessitou de fixação por parafuso, conforme mostra a Figura 70.

Figura 70: Montagem dos sensores HC-SR04 e placa Arduino UNO



Fonte: O Autor (2022).

Já o Kinect foi fixado diretamente na tampa para não cair no deslocamento do robô, conforme Figura 71.

Figura 71: Montagem Kinect na tampa.



Fonte: O Autor (2022).

Por fim, fixado o suporte octogonal no chassi do robô (Figura 72). A versatilidade em utilizar o perfil de alumínio, permite que o suporte possa ser deslocado longitudinalmente para melhor ajuste do centro de massa do robô.

Figura 72: Fixação do suporte octogonal no chassi do robô Tupy.



Fonte: O Autor (2022).

3.1.3.2 Eletrônica

A etapa eletrônica prezou-se pela ligação dos componentes eletrônicos (Arduino, sonares, RPi, Ponte H dupla) e montagem do circuito da bateria.

- Ligação dos sensores ultrassônicos.

Para ligação dos sensores ultrassônicos foram utilizados cabos *flat* macho-fêmea. Demonstrativamente, foi ligado apenas de 4 sonares (frontal, traseiro, esquerdo e direito) e optado por enclausurar o Arduino no suporte octogonal, por questões de segurança e organização das conexões elétricas. A Figura 73, mostra como ficou a disposição destes componentes.

Figura 73: Ligação dos sonares no suporte octogonal



Fonte: O Autor (2022).

- Ligação placa RPi 3b+.

Em seguida foram conectados todos os cabos USBs dos periféricos e alimentação micro USB no RPi. A Figura 74 demonstra essa situação.

Figura 74: Ligação elétrica placa RPi

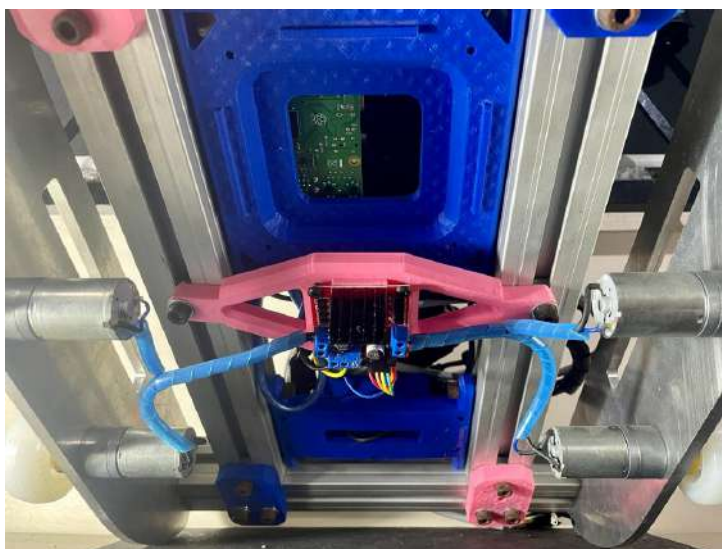


Fonte: O Autor (2022).

- Ligação ponte H.

A ponte H dupla L298N ficou acoplada na parte inferior do chassi (Figura 75), ligando os dois motores. Já a ligação do controle é feita por um cabo *flat* de 5 vias: 4 para controle e 1 para referência (GND). Esse cabo vai para a placa Arduino, localizado no suporte octogonal.

Figura 75: Integração: Arduino — Sonares (HC-SR04).



Fonte: O Autor (2022).

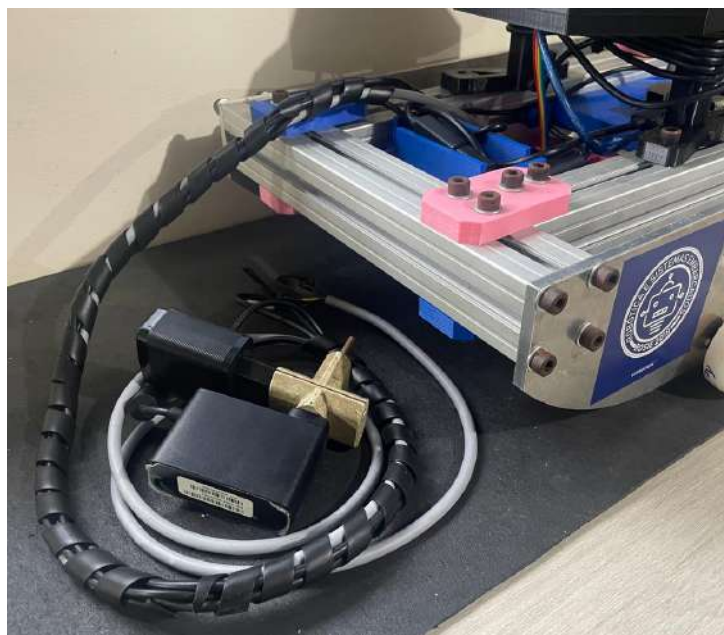
- Ligação da alimentação.

O robô pode ser alimentado de duas formas: por fonte ou bateria. Ambas devem conseguir alimentar todos os periféricos como, Kinect (12V-1A máx.), RPi (5V-2,5A máx.) e Ponte H (5V-4A máx.)

Como a integração demandou várias horas de testes, foi conveniente optar pela alimentação vinda da fonte, conforme mostrado na Figura 76.

Já bateria não foi utilizada, mas fica tem seu local definido na parte traseira do robô. Porém, no caso da bateria, necessita-se preparar um circuito específico para cada alimentação.

Figura 76: Fonte.



Fonte: O Autor (2022).

- Montagem final.

Por fim, após a montagem de todos os componentes, tem-se o robô Tupy montado no ambiente físico, conforme Figura 77.

Figura 77: Integração: Arduino — Sonares (HC-SR04).



Fonte: O Autor (2022).

Tabela 6: 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 (2022).

Desta forma, para que o ROS Melodic funcione corretamente e sem surpresas, conforme Tabela 6, a distribuição a ser instalada no RPi (armhf) e *desktop* (amd64) será o **Linux Ubuntu 18.04 LTS Bionic Beaver**.

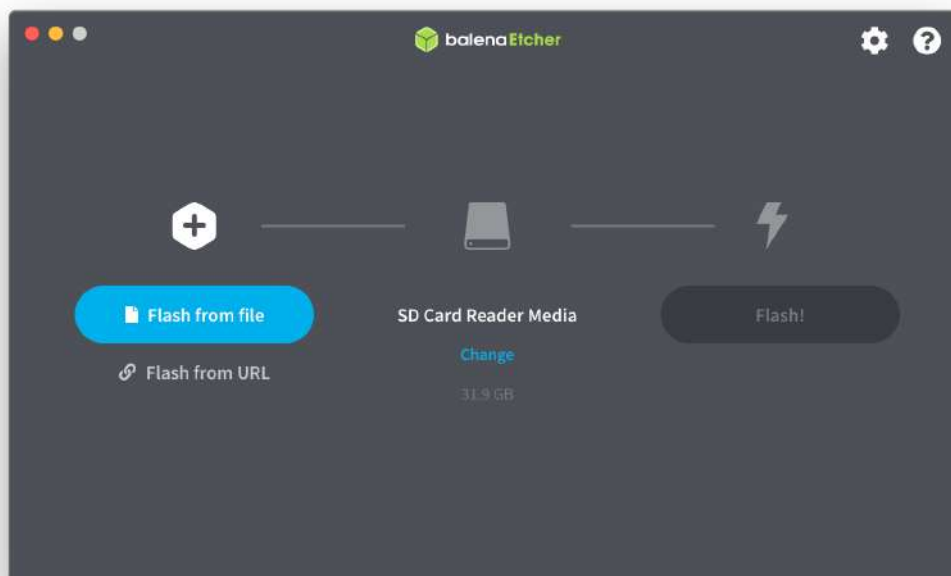
3.2.2 Instalação do SO no *desktop*: Arquitetura 64 bits (amd64)

No *desktop*, foi realizado *dual-boot* para a instalação do **Linux Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-131-generic x86_64)** lado a lado com o Windows 10. O principal motivo por instalar fisicamente o sistema operacional, foi pelo fato do melhor desempenho nas tarefas de visualização das imagens, simulação e comunicação USB sem problemas de compatibilidade entre periféricos. Visto que, inicialmente, foram feitos vários testes com a máquina virtual para entender melhor o funcionamento do ROS e integração entre dispositivos, entretanto as condições estabelecidas virtualmente não foram suficientes atender as expectativas do projeto.

3.2.3 Instalação do SO no RPi 3B+: Arquitetura 32 bits (armhf)

No RPi 3B+ também foi instalado a distribuição **18.04.2 LTS (GNU/Linux 4.15.0-1033-raspi2 armv7l) para Arquitetura 32 Bits (Armhf)**. Para realizar a instalação do sistema operacional, basta carregar a imagem do Ubuntu na plataforma Balena Etcher (ferramenta que permite gravar sistemas operacionais em um dispositivo de armazenamento móvel), selecionar o cartão *Secure Digital* – Secure Digital (SD) e realizar a gravação do SO, conforme Figura 79.

Figura 79: Gravação na plataforma Balena Etcher.



Fonte: O Autor (2022).

Após a gravação no cartão, basta inseri-lo no *slot* para cartão micro SD (Figura 80) e o Ubuntu irá carregar no RPi.

Figura 80: Cartão Micro SD inserido no RPi.



Fonte: O Autor (2022).

3.2.4 Interface de comunicação *desktop* com RPi

Feito as instalações básicas dos sistemas operacionais nos dispositivos, passa-se a configurá-los em rede para ser possível estabelecer uma conexão segura, para troca de informações e dados via rede. Para isso utilizaremos a Internet de Serviços — Protocolo TCP/IP. Essa conexão será estabelecida através do Wi-Fi (IEEE 802.11) em uma Rede Local. Com isso podemos utilizar o SSH em ambas estações para acesso remoto, conforme configuração a seguir:

3.2.4.1 Endereçamento IP dos dispositivos

Vale lembrar que as informações a seguir foram definidas conforme configurações da rede local, podendo ser alteradas quando necessárias. Desta maneira, define-se a máscara de rede padrão (255.255.255.0) e em seguida os endereços *Internet Protocol Version 4* – Protocolo de Internet Versão 4 (IPv4):

- **192.168.15.21 — *desktop***
- **192.168.15.22 — RPi 3B+**

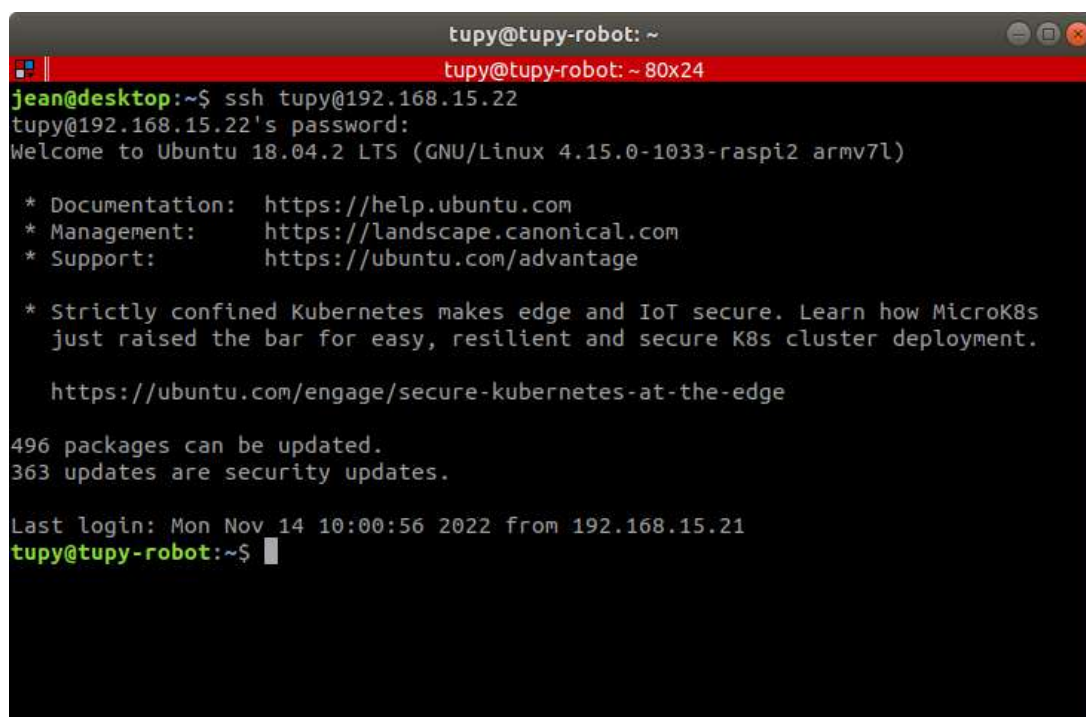
3.2.4.2 Acesso remoto

O protocolo SSH será utilizado para o acessar remotamente o robô Tupy. Ele permite estabelecer uma conexão segura entre máquinas Linux através do terminal. Após habilitar este serviço nas estações, pode-se, então, acessá-los remotamente através dos seguintes endereços dados aos dispositivos:

- **jean@192.168.15.21 — *desktop***
- **tupy@192.168.15.22 — RPi 3B+**

A partir de agora, já é possível trabalhar com o Raspberry remotamente, sem a necessidade da utilização de teclado, mouse e monitor. Para acessar o Raspberry via *desktop* basta acessar o terminal e digitar o seguinte comando: **ssh tupy@192.168.15.22** e a senha, conforme definido na instalação do sistema, conforme o exemplo da Figura 81:

Figura 81: Conexão SSH do *desktop* para o robô Tupy.



```
tupy@tupy-robot: ~
tupy@tupy-robot: ~ 80x24
jean@desktop:~$ ssh tupy@192.168.15.22
tupy@192.168.15.22's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-1033-raspi2 armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

496 packages can be updated.
363 updates are security updates.

Last login: Mon Nov 14 10:00:56 2022 from 192.168.15.21
tupy@tupy-robot:~$
```

Fonte: O Autor (2022).

3.2.5 Instalação do Framework ROS Melodic

3.2.5.1 *desktop*: ros-melodic-desktop-full

Já o *Framework* ROS, teve a sua versão completa instalada. Esta versão traz todos os recursos da plataforma, ferramentas de visualização e simuladores 2D e 3D, como o *ROS Vizualization* – Visualização ROS (Rviz) e Gazebo, por exemplo.

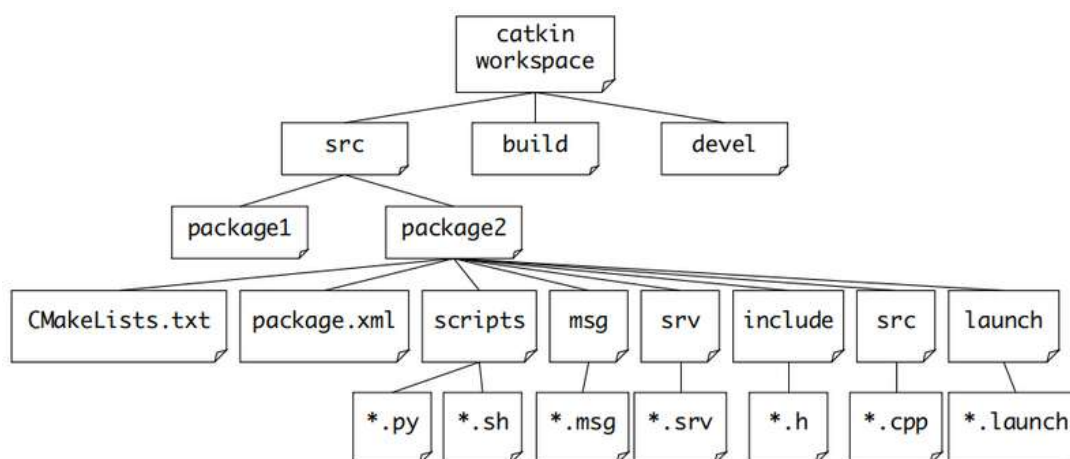
3.2.5.2 RPi 3B+: ros-melodic-base

Através do uso do serviço SSH, podemos realizar todas as configurações necessárias no *RPi* via *desktop*. Com essa condição, instalou-se o **ros-melodic-base** no microcontrolador. Como o *RPi* tem uma capacidade de processamento limitada comparada ao *desktop*, preferiu-se instalar a versão básica do ROS Melodic. Esta versão traz somente os recursos necessários para o *Framework* funcionar.

3.2.6 Criação de *workspace catkin*

Para cada dispositivo será necessário criar um *workspace*. Esse ambiente é um diretório no Linux, conforme mostrado na Figura 82, necessário para armazenar todos os pacotes desenvolvidos para o ROS.

Figura 82: Estrutura de Arquivos do *catkin workspace*.



Fonte: Adaptado de *Develop PAPER* (2022).

No caso do *desktop*, por exemplo, foi instalado no *workspace catkin* alguns pacotes do para testes para o robô Tupy, conforme mostra a tela do terminal na Figura 83.

Figura 83: Estrutura de Arquivos do *catkin workspace* do Tupy.

```

jean@desktop: ~/catkin_ws/src
jean@desktop: ~/catkin_ws/src 80x10
jean@desktop:~$ cd catkin_ws
jean@desktop:~/catkin_ws$ ls
build  devel  install  src
jean@desktop:~/catkin_ws$ cd src/
jean@desktop:~/catkin_ws/src$ ls
CMakeLists.txt  img_processor  octomap_msgs  roserial      sonar
find-object     navigation    pcl_msgs      rtabnap_ros
jean@desktop:~/catkin_ws/src$
  
```

Fonte: O Autor (2022).

3.2.7 Integração Virtual — Arduino com ROS

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, também, 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 Inscrever mensagens.

3.2.7.1 Comunicação Arduino com ROS

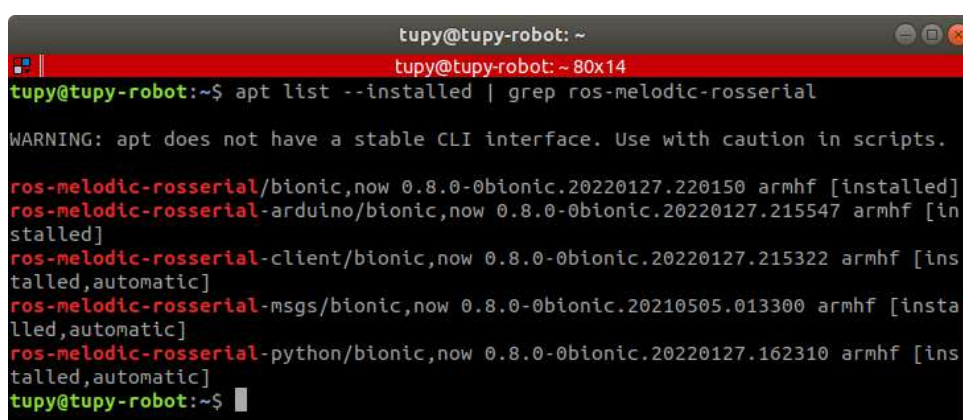
Para associar o Arduino ao ROS é indispensável o uso de determinados pacotes. O principal deles é o `rosterial-arduino`, nele estabelece a comunicação serial RX/TX, tornando-se um ‘meio’ para a troca de mensagens. A partir de então será possível que o ROS reconheça, por exemplo, os Tópicos dos sonares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

Partindo para a etapa de configuração da comunicação, devido aos testes e gravações realizadas com a IDE Arduino e a necessidade da interface USB para trocar informações, a instalação dos seguintes pacotes foram feitas em ambas as plataformas (*desktop* e RPi), digitando os comandos no *terminal*:

- **`sudo apt-get install ros-melodic-rosterial`**
- **`sudo apt-get install ros-melodic-rosterial-arduino`**

Para conferir que os pacotes foram instalados podemos executar o seguinte comando no terminal (Figura 84): **`apt list --installed | grep ros-melodic-rosterial`**

Figura 84: Pacotes Rossterial Instalados.



```
tupy@tupy-robot: ~
tupy@tupy-robot: ~ 80x14
tupy@tupy-robot:~$ apt list --installed | grep ros-melodic-rosterial

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

ros-melodic-rosterial/bionic,now 0.8.0-0bionic.20220127.220150 armhf [installed]
ros-melodic-rosterial-arduino/bionic,now 0.8.0-0bionic.20220127.215547 armhf [installed]
ros-melodic-rosterial-client/bionic,now 0.8.0-0bionic.20220127.215322 armhf [installed,automatic]
ros-melodic-rosterial-msgs/bionic,now 0.8.0-0bionic.20210505.013300 armhf [installed,automatic]
ros-melodic-rosterial-python/bionic,now 0.8.0-0bionic.20220127.162310 armhf [installed,automatic]
tupy@tupy-robot:~$
```

Fonte: O Autor (2022).

Feito isto, passa-se a realizar as configurações na interface de desenvolvimento do Arduino. Para isso, é imprescindível a inclusão da **Rossterial Arduino**

Library 0.7.9 no Gerenciador de Bibliotecas Arduino IDE. Com a biblioteca instalada, basta incluir no cabeçalho da programação a função ROS que melhor se encaixe na programação.

Conforme a necessidade do projeto, também foram incluídas outras duas bibliotecas. Na sequência (Figura 85), seguem todas as bibliotecas adicionais instaladas no Gerenciador de Bibliotecas Arduino IDE:

Figura 85: Gerenciador de Biblioteca Arduino IDE.



Fonte: O Autor (2022).

Já na Figura 86, nota-se como se dera a declaração destas funções no cabeçalho de programação principal na IDE Arduino.

Figura 86: Biblioteca Utilizadas na Programação do Arduino.

```
#define IN1 10 //define que o pino 10 receberá o valor da variável IN1
#define IN2 11 //define que o pino 11 receberá o valor da variável IN2
#define IN3 12 //define que o pino 12 receberá o valor da variável IN3
#define IN4 13 //define que o pino 13 receberá o valor da variável IN4

#define SONAR_NUM 4           //Número de sonares
#define MAX_DISTANCE 200     //Maxima distância de detecção dos obstáculos
#define PING_INTERVAL 33     //Loop do ping a cada 33 microsegundos.
```

Fonte: O Autor (2022).

Para segmentar melhor cada aplicação, serão apresentados a seguir, a programação individual tanto dos motores quanto dos sonares. A versão final do programa com a mescla dos dois códigos estará disponível no Apêndice D.

3.2.7.2 Controle Ponte H Motores (*Subscriber* ROS)

O **Twist** é o pacote que envia mensagens geométricas *geometry_msgs* que fornece mensagens primitivas geométricas comuns, como pontos, vetores (lineares/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 tele operado 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**

Já a preparação da programação do Arduino deve ser feita com base na inscrição deste tópico. Então define-se no cabeçalho da IDE algumas informações pertinente a isto. A Figura 87 demonstra como se deu a inclusão das funções da biblioteca *roserial-arduino* e da declaração dos pinos dos motores conforme ligação elétrica.

Figura 87: Definição iniciais da Ponte H na IDE Arduino.

```
#include <ros.h> //dedicada à comunicação dos pacotes roserial-arduino
#include <ros/time.h> //responsável pela sincronização dos tempos roserial-arduino
#include <geometry_msgs/Twist.h> //trata dos vetores via roserial-arduino
#include <sensor_msgs/Range.h> //trata dos dados de distância roserial-arduino

#define IN1 10 //define que o pino 10 receberá o valor da variável IN1
#define IN2 11 //define que o pino 11 receberá o valor da variável IN2
#define IN3 12 //define que o pino 12 receberá o valor da variável IN3
#define IN4 13 //define que o pino 13 receberá o valor da variável IN4
```

Fonte: O Autor (2022).

Feito isso, parte-se para a preparação da lógica de programação para o funcionamento dos motores em Ponte H. Chama-se a Função '**void onTwist**' receber, através das **geometry_msgs/Twist.h**, os valores vetoriais para acionar as saídas do Arduino. Após a criação desta função, deve-se indicar que o Arduino irá subcrever o tópico **cmd_vel**, ou seja, estará apto a receber as informações de teleoperação envidas pelo *desktop*, como mostrado na Figura 88.

Figura 88: Programação básica para teleoperação com ponte H.

```

void onTwist(const geometry_msgs::Twist& msg)
{
    if(msg.linear.x > 0) //TUPY PARA FRENTE
    {
        digitalWrite(IN1,HIGH); //roda direita para frente ON
        digitalWrite(IN2,LOW); //roda direita para trás OFF
        digitalWrite(IN3,HIGH); //roda esquerda para frente ON
        digitalWrite(IN4,LOW); //roda esquerda para trás OFF
    }
    else if(msg.linear.x < 0) //TUPY PARA TRÁS
    {
        digitalWrite(IN1,LOW); //roda direita para frente OFF
        digitalWrite(IN2,HIGH); //roda direita para trás ON
        digitalWrite(IN3,LOW); //roda esquerda para frente OFF
        digitalWrite(IN4,HIGH); //roda esquerda para trás ON
    }
    else if(msg.angular.z < 0) //TUPY GIRA PARA ESQUERDA
    {
        digitalWrite(IN1,HIGH); //roda direita para frente ON
        digitalWrite(IN2,LOW); //roda direita para trás OFF
        digitalWrite(IN3,LOW); //roda esquerda para frente OFF
        digitalWrite(IN4,HIGH); //roda esquerda para trás ON
    }
    else if(msg.angular.z > 0) //TUPY GIRA PARA DIREITA
    {
        digitalWrite(IN1,LOW); //roda direita para frente OFF
        digitalWrite(IN2,HIGH); //roda direita para trás ON
        digitalWrite(IN3,HIGH); //roda esquerda para frente ON
        digitalWrite(IN4,LOW); //roda esquerda para trás OFF
    }
    else //PARAR TUPY
    {
        digitalWrite(IN1,LOW); //roda direita para frente OFF
        digitalWrite(IN2,LOW); //roda direita para trás OFF
        digitalWrite(IN3,LOW); //roda esquerda para frente OFF
        digitalWrite(IN4,LOW); //roda esquerda para trás OFF
    }
}
//Inscreve objetos do referentes aos comandos da teleoperação
ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel",onTwist);
//cria um objeto que representa o nó ROS.
ros::NodeHandle nh;

```

Fonte: O Autor (2022).

Por fim, configuram-se os pinos como saídas digitais, a inicialização do nó ROS, que a programação irá inscrever informações ROS e o processamento em *loop* do retorno dos Tópicos inscritos (Figura 89).

Figura 89: Configuração final da ponte H na IDE Arduino.

```

void setup() // código de configuração para executar uma vez:
{
    pinMode(IN1, OUTPUT); //Configura a variável IN1 (pin 10) como saída digital
    pinMode(IN2, OUTPUT); //Configura a variável IN2 (pin 11) como saída digital
    pinMode(IN3, OUTPUT); //Configura a variável IN3 (pin 12) como saída digital
    pinMode(IN4, OUTPUT); //Configura a variável IN4 (pin 13) como saída digital

    nh.initNode(); //inicia o nó ROS para o processo
    nh.subscribe(sub); //define que a programação irá inscrição do nó
}

void loop() //código principal para ser executado repetidamente:
{
    nh.spinOnce(); //O ROS processa o retorno dos tópicos inscritos
}

```

Fonte: O Autor (2022).

No Apêndice B poderá ser verificado a programação individual completa da ponte H realizada na IDE Arduino.

3.2.7.3 Sensores Ultrassônicos (*Publisher* ROS)

Assim como as mensagens enviadas pelo pacote Twist, temos o pacote Range. Ele é responsável por enviar mensagens de distância, no caso, o valor da leitura dos sensores ultrassônicos. Entretanto, não temos um pacote ROS criado para este fim. Portanto, será criado os Nós ROS dos quatro sensores na placa Arduino UNO para ser possível visualizar as leituras de distância de cada um deles. Nesta etapa será apresentado, tal qual o Controle da Ponte H, o funcionamento em partes do programa na Arduino IDE.

Figura 90: Cabeçalho Sensores Ultrassônicos na IDE Arduino.

```
#include <ros.h> //dedicada à comunicação dos pacotes ROS
#include <ros/time.h> //responsável pela sincronização dos tempos ROS
#include <sensor_msgs/Range.h> //trata do tipo de mensagem ROS
#include <NewPing.h> //biblioteca dedicada ao uso de sonares
#include <SimpleKalmanFilter.h> //biblioteca de filtros digitais

#define SONAR_NUM 4 //Número de sonares
#define MAX_DISTANCE 200 //Maxima distância de detecção dos obstáculos
#define PING_INTERVAL 33 //Loop do ping a cada 33 microsegundos.

unsigned long pingTimer[SONAR_NUM]; //tempo de ping para cada sensor.
unsigned int cm[SONAR_NUM]; // Onde as distâncias de ping são armazenadas.
uint8_t currentSensor = 0; // Mantém o controle de qual sensor está ativo.
unsigned long _timerStart = 0; //tempo inicia em 0
int LOOPING = 40; //Loop a cada 40 milissegundos.
uint8_t oldSensorReading[3]; //Armazena o último valor válido dos sensores.

uint8_t oneSensor; //Armazena o valor real do sensor 1.
uint8_t twoSensor; //Armazena o valor real do sensor 2.
uint8_t threeSensor; //Armazena o valor real do sensor 3.
uint8_t fourSensor; //Armazena o valor real do sensor 4.

uint8_t oneSensorKalman; //Armazena o valor filtrado do sensor 1.
uint8_t twoSensorKalman; //Armazena o valor filtrado do sensor 2.
uint8_t threeSensorKalman; //Armazena o valor filtrado do sensor 3.
uint8_t fourSensorKalman; //Armazena o valor filtrado do sensor 4.

NewPing sonar[SONAR_NUM] = //cria objetos newPing para todos os sensores.
{
    NewPing(3, 2, MAX_DISTANCE), //Pino de Trigger, Pino de echo, distância máxima.
    NewPing(5, 4, MAX_DISTANCE),
    NewPing(7, 6, MAX_DISTANCE),
    NewPing(9, 8, MAX_DISTANCE)
};

//criar objetos de filtro Kalman para os sensores.
SimpleKalmanFilter KF_1(2, 2, 0.01); //(incerteza medição, incerteza estimada, ruído);
SimpleKalmanFilter KF_2(2, 2, 0.01);
SimpleKalmanFilter KF_3(2, 2, 0.01);
SimpleKalmanFilter KF_4(2, 2, 0.01);

ros::NodeHandle nh; //iniciará o nó na placa Arduino.
```

Fonte: O Autor (2022).

Começando a programação na IDE Arduino, declaram-se as bibliotecas e funções, algumas definições e variáveis no cabeçalho (Figura 90). 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. Posteriormente, criam-se objetos **NewPing**

e **SimpleKalmanFilter** para todos os sensores conforme a indicação dos parâmetros comentadas no programa. Já **NodeHandle** é um objeto que representa que a placa será um nó do ROS.

Figura 91: Função sensorCycle ().

```
void sensorCycle() //loop em todos os sensores
{
    for (uint8_t i = 0; i < SONAR_NUM; i++) {
        if (millis() >= pingTimer[i]) {
            pingTimer[i] += PING_INTERVAL * SONAR_NUM;
            if (i == 0 && currentSensor == SONAR_NUM - 1) oneSensorCycle();
            sonar[currentSensor].timer_stop();
            currentSensor = i;
            cm[currentSensor] = 0;
            sonar[currentSensor].ping_timer(echoCheck);
        }
    }
}
```

Fonte: O Autor (2022).

Em seguida, chama-se a função **sensorCycle()**, ela faz um loop por todos os sensores e, quando o ciclo de *ping* do sensor estiver concluído, adiciona os resultados na função **oneSensorCycle()**, conforme Figura 91.

Figura 92: Função echoCheck.

```
void echoCheck() // Se o ping for recebido, defina a distância do sensor para array.
{
    if (sonar[currentSensor].check_timer())
        cm[currentSensor] = sonar[currentSensor].ping_result / US_ROUNDTRIP_CM;
}
```

Fonte: O Autor (2022).

Quando a função **echoCheck()** recebe um novo *ping*, o valor da distância do sensor é adicionada a um *array*, conforme Figura 92.

Figura 93: Função oneSensor.

```
void oneSensorCycle() // Retorna o último valor válido do sensor.
{
    oneSensor = returnLastValidRead(0, cm[0]);
    twoSensor = returnLastValidRead(1, cm[1]);
    threeSensor = returnLastValidRead(2, cm[2]);
    fourSensor = returnLastValidRead(3, cm[3]);
}
```

Fonte: O Autor (2022).

Já a função **oneSensorCycle()** (Figura 93), retorna o último valor válido recebido do sensor.

Figura 94: Função returnLastValidRead.

```
int returnLastValidRead(uint8_t sensorArray, uint8_t cm)
//Se o valor do sensor for 0, então retorna o último valor armazenado diferente de 0.
{
    if (cm != 0) {
        return oldSensorReading[sensorArray] = cm;
    } else {
        return oldSensorReading[sensorArray];
    }
}
```

Fonte: O Autor (2022).

Na **returnLastValidRead**, filtram-se as leituras do sensor, aloca as leituras diferentes de 0. Caso a leitura for 0, retorna-se a última saída válida. Desta forma, as leituras falsas do sensor ultrassônico são ignoradas. Conforme lógica da Figura 93.

Figura 95: Função applyKF.

```
void applyKF() //Aplica o Filtro Kalman na leitura do sensor.
{
    oneSensorKalman    = KF_1.updateEstimate(oneSensor);
    twoSensorKalman    = KF_2.updateEstimate(twoSensor);
    threeSensorKalman  = KF_3.updateEstimate(threeSensor);
    fourSensorKalman   = KF_4.updateEstimate(fourSensor);
}
```

Fonte: O Autor (2022).

Neste próximo bloco, aplica-se o filtro digital de Kalman **applyFK()** (Figura 94), na leitura do sensor.

Figura 96: Funções de Tempo.

```
void startTimer() //a função para começar a contar o tempo usando millis()
{
    _timerStart = millis();
}

bool isTimeForLoop(int _mSec) //Verifica se o tempo passou e retorna true.
{
    return (millis() - _timerStart) > _mSec;
}
```

Fonte: O Autor (2022).

Essas funções referem-se a contagem do tempo. É utilizado **millis()**, por ser mais preciso que a função **delay()**. A **startTimer()**, inicia a contagem do tempo. Já a **isTimeForLoop()**, verifica se o tempo passou, retornando *true*.

Figura 97: Escrevendo mensagens para o ROS.

```
void sensor_msg_init(sensor_msgs::Range &range_name, char *frame_id_name)
{
    range_name.radiation_type = sensor_msgs::Range::ULTRASOUND;
    range_name.header.frame_id = frame_id_name;
    range_name.field_of_view = 0.26;
    range_name.min_range = 0.0;
    range_name.max_range = 2.0;
}

//Cria instâncias para mensagens de distância.
sensor_msgs::Range range_1;
sensor_msgs::Range range_2;
sensor_msgs::Range range_3;
sensor_msgs::Range range_4;

//Cria objetos ROS de todos os sensores para publicação
ros::Publisher pub_range_1("/sonar1", &range_1);
ros::Publisher pub_range_2("/sonar2", &range_2);
ros::Publisher pub_range_3("/sonar3", &range_3);
ros::Publisher pub_range_4("/sonar4", &range_4);
```

Fonte: O Autor (2022).

Nesta etapa são realizadas as configurações da mensagem **Range**.

- O primeiro parâmetro é o tipo do sensor, logo definido como ULTRASOUND;
- O frame_id é usado para especificar o ponto de referência para os dados contidos nessa mensagem;
- O field_of_view representa o tamanho do ângulo de leitura, será definido 0,26 radianos que equivalem a 15 graus;
- min_range refere-se a distância mínima de leitura;
- max_range refere-se a distância máxima de leitura;

Então, define-se quatro objetos do tipo *Range* e o nome dos Tópicos **/sonar1**, **/sonar2**, **/sonar3** e **/sonar4**.

Figura 98: setup().

```

void setup()
{
    pingTimer[0] = millis() + 75;
    for (uint8_t i = 1; i < SONAR_NUM; i++)
        pingTimer[i] = pingTimer[i - 1] + PING_INTERVAL;

    nh.initNode(); //inicia o nó ROS para o processo
    nh.advertise(pub_range_1);
    nh.advertise(pub_range_2);
    nh.advertise(pub_range_3);
    nh.advertise(pub_range_4);

    sensor_msg_init(range_1, "/sonar1");
    sensor_msg_init(range_2, "/sonar2");
    sensor_msg_init(range_3, "/sonar3");
    sensor_msg_init(range_4, "/sonar4");
}

```

Fonte: O Autor (2022).

O comando **nh.advertise()** é utilizado para criar um *Publisher* com a leitura dos Tópicos.

Figura 99: loop().

```

void loop() {
    if (isTimeForLoop(LOOPING)) {
        sensorCycle();
        oneSensorCycle();
        applyKF();
        range_1.range = oneSensorKalman;
        range_2.range = twoSensorKalman;
        range_3.range = threeSensorKalman;
        range_4.range = fourSensorKalman;

        range_1.header.stamp = nh.now();
        range_2.header.stamp = nh.now();
        range_3.header.stamp = nh.now();
        range_4.header.stamp = nh.now();

        pub_range_1.publish(&range_1);
        pub_range_2.publish(&range_2);
        pub_range_3.publish(&range_3);
        pub_range_4.publish(&range_4);

        startTimer();
    }
    nh.spinOnce(); //Informa ao ROS que uma nova mensagem chegou.
}

```

Fonte: O Autor (2022).

Na função **loop()**, verifica-se o tempo para publicar a cada 40ms. Em seguida, é lido o valor retornado do sensor e com a **nh.now()**, retorna-se o tempo atual e publica o valor da distância.

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.

Portanto, demonstrou-se como se deu a integração virtual dos sonares com o ROS. Este também estará completo, no Apêndice C. Lembrando que o programa final (Apêndice D) do robô Tupy do Arduino, foi mesclado os dois programas anteriores e pronto para receber oito sonares, conforme ideia inicial do projeto.

3.2.7.4 Carregando o *sketch* no microcontrolador

3.2.8 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 Nós 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.

3.2.8.1 Instalando drivers Kinect

Existem alguns drivers disponíveis para o Kinect, porém somente dois deles são dedicados ao uso com o ROS. Foram utilizados os dois drivers listados abaixo, cada um para um dispositivo.

- Drivers OpenNI — (https://github.com/ros-drivers/opensni_camera)
- Drivers 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 no *desktop* se deu para fins de testes, ou seja, não sendo necessário para a aplicação final do Gêmeo Digital com o robô Tupy. Já no RPi, o **Libfreenect** foi instalado 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, os drivers libfreenect foram construídos 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. A seguir seguem os códigos utilizados para iniciar o procedimento:

Figura 100: Códigos para instalação dos drivers do Kinect.



```

sudo apt-get update
sudo apt-get install cmake build-essential libusb-1.0-0-dev
git clone https://github.com/OpenKinect/libfreenect.git
cd libfreenect
mkdir build && cd build
cmake -L ..
make
sudo make install

cd catkin

```

sh ▾ Largura da tabulação: 8 ▾ Lin 19, Col 1 ▾ INS

Fonte: O Autor (2022).

O próximo passo foi instalar o pacote **freenect_stack** para ROS que fornece uma interface ROS para o Microsoft Kinect usando a biblioteca **libfreenect**. Para isso clonou-se o repositório dessa biblioteca, localizada no 'Git-Hub', na pasta '*catkin_ws/src*' e, em seguida, compilados com comando '*catkin_make*'.

Figura 101: Códigos Git-Clone para drivers do Kinect.



```

cd ~/catkin_ws/src

git clone https://github.com/ros-drivers/freenect_stack.git
git clone https://github.com/ros-perception/image_common.git
git clone https://github.com/ros-drivers/rgbd_launch.git
git clone https://github.com/ros-perception/vision_opencv.git
git clone https://github.com/ros-perception/image_pipeline.git
git clone https://github.com/ros/geometry2.git

cd ..

sudo apt-get install libbullet-dev libharfbuzz-dev libgtk2.0-dev libgtk-3-dev
catkin_make -j2

```

sh ▾ Largura da tabulação: 8 ▾ Lin 22, Col 16 ▾ INS

Fonte: O Autor (2022).

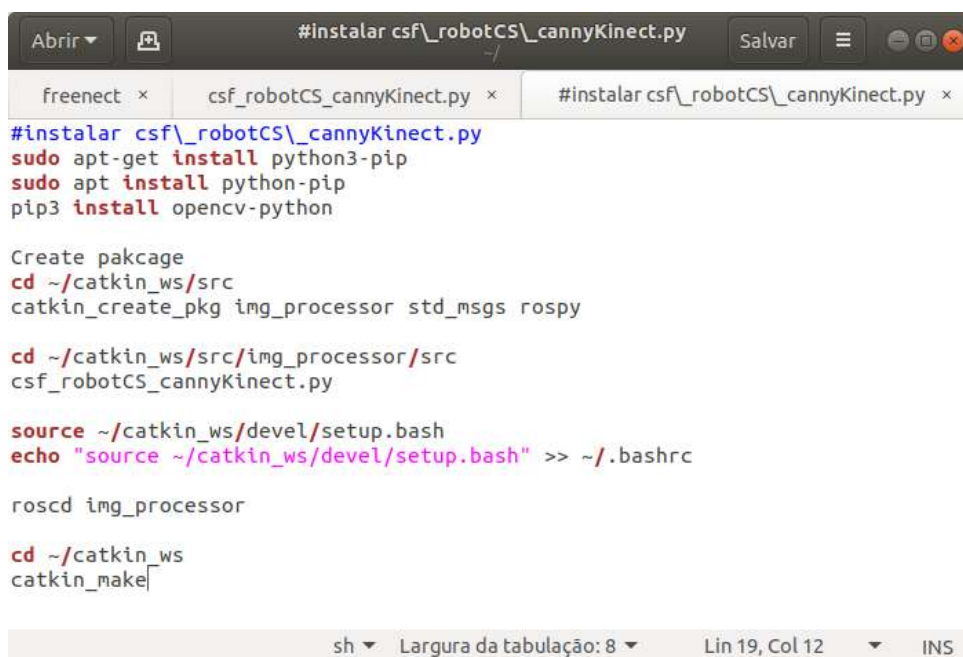
3.2.8.2 Instalando OpenCV Kinect no RPi

Para a instalação do **OpenCV**, buscaram-se referências do código utilizado por Magrin, Del Conte e Todt (2021). Para isso, clonou-se o arquivo escrito em Python, disponível no 'Git-Hub' deste autor:

- **csf_robotCS_cannyKinect.py**

A instalação foi realizada no *desktop*. Em seguida, para o ROS reconhecer este arquivo, foi necessário transformá-lo em um pacote ROS, conforme os comandos (Figura 102).

Figura 102: Códigos para criar o pacote Canny Kinect.



```

#instalar csf_robotCS_cannyKinect.py
sudo apt-get install python3-pip
sudo apt install python-pip
pip3 install opencv-python

Create package
cd ~/catkin_ws/src
catkin_create_pkg img_processor std_msgs rospy

cd ~/catkin_ws/src/img_processor/src
csf_robotCS_cannyKinect.py

source ~/catkin_ws/devel/setup.bash
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc

roscd img_processor

cd ~/catkin_ws
catkin_make
  
```

Fonte: O Autor (2022).

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. Para isto, foi aberto o arquivo na pasta **/home/jean/catkin_ws/src/img_processor/src** e editado os seguintes itens sublinhados em vermelho da Figura 103. O arquivo final estará disponível no Apêndice E.

Figura 103: Configuração do pacote Canny Kinect.

```
#callback function funcao principal
def img_callback(ros_data):
    global delta
    print(int(round(time.time()*1000)) - delta)
    delta = int(round(time.time()*1000))

    try:
        image_np = bridge.imgmsg_to_cv2(ros_data, "mono8") #tipagem do open cv rgb
        #image_np = bridge.imgmsg_to_cv2(ros_data, "bgr8") #tipagem do open cv em escala de cinza
        flipVertical_image_np = cv2.flip(image_np, 1)
    except CvBridgeError as e:
        print(e)

    edges = cv2.Canny(image_np, 50, 150, apertureSize=3) #basicamente um novo frame. opera em escala de cinza
    flipVertical_edges = cv2.flip(edges, 1)

    #cv2.imshow("output", image_np)
    #cv2.imshow("output", np.hstack([image_np, edges])) #concatenacao de matrizes
    cv2.imshow("output", np.hstack([flipVertical_image_np, flipVertical_edges]))
    cv2.waitKey(1) #sleep de 1ms

def main():
    rospy.init_node('csf_robotCS_cannyKinect')

    rospy.Subscriber("camera/rgb/image_raw", Image, img_callback, queue_size=1)
    #rospy.Subscriber("/kinect/rgb_image", Image, img_callback, queue_size=1)

    try:
        rospy.spin()
    except KeyboardInterrupt:
        print('Shutting down ...')
    cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```

Fonte: Adaptado de (MAGRIN; DEL CONTE; TODT, 2021).

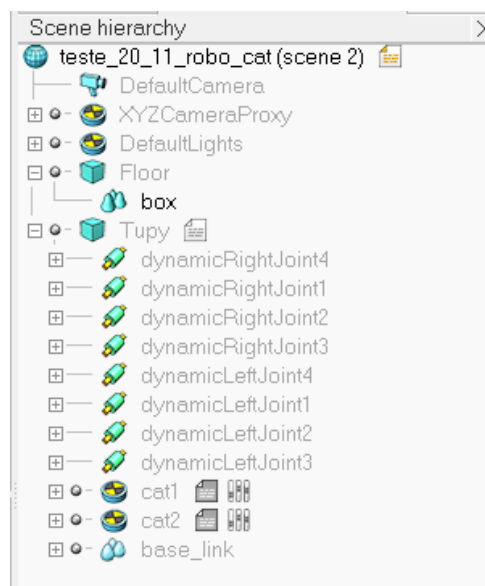
3.2.9 Integração Virtual — ROS com CoppeliaSim

O CoppeliaSim, antigo V-REP, permite a simulação virtual deste projeto. Esse *software* traz diversos recursos interessantes nesse sentido. O grande benefício disto, se dera pelo fato de permitir a interconexão com o ambiente ROS. Como esse estudo baseou-se no artigo de Magrin, Del Conte e Todt (2021), utilizou-se da modelagem base do robô Tupy, disponível no repositório 'Git-Hub', 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.

Na Árvore do Projeto, nota-se que é possível manipular individualmente cada peça, devido o CAD do robô vir em extensão (.STL). Apenas pequenos ajustes na ordem de cada peça na camada foram necessários para simulação.

Figura 104: Árvore projeto CoppeliaSim.



Fonte: O Autor (2022).

Para a movimentação controlada por teleoperação, necessitou-se inscrever o tópico **cmd_vel** publicado por **teleop_twist_key**. A seguir, será demonstrado como ocorreu a interface de comunicação do ROS com a simulação do robô, configurando o *scripts* **LUA** de programação principal do robô. Da mesma forma que ocorreu a explicação da interface realizada pela Arduino IDE, será exibido cada bloco de função de forma comentada.

Figura 105: sysCall_init.

Fonte: O Autor (2022).

A função **sysCall_init()**, será executado uma vez logo no início de uma simulação. Ela define algumas condições e parâmetros como:

- Localização da conexão com a interface ROS. Caso verdadeiro, inscreve-se o tópico **cmd_vel** ;
- Interação do movimento das rodas (esteiras);
- Parâmetros iniciais zerados;
- Distância entre eixo das rodas (esteiras);

Figura 106: sysCall_actuation.

```
function sysCall_actuation()

    function Message_callback(vel)

        leftVelocity= vel.linear.x - d*vel.angular.z
        rightVelocity= vel.linear.x + d*vel.angular.z

        -- Make the tracks rotate (only visual effect):
        sim.writeCustomTableData(cat1,"_ctrl1_",{vel=rightVelocity})
        sim.writeCustomTableData(cat2,"_ctrl1_",{vel=leftVelocity})

        for i=1,4,1 do
            sim.setJointTargetVelocity(leftDynamicMotors[i],-leftVelocity/(wheelRadius))
            sim.setJointTargetVelocity(rightDynamicMotors[i],-rightVelocity/(wheelRadius))
            print(vel.linear.x)
        end

    end

end
```

Fonte: O Autor (2022).

A **sysCall_actuation()**, retorna a chamada que será executada em *loop*, responsável por lidar com todas as funcionalidades de atuação da simulação. Nesse bloco, define-se a lógica para a movimentação das rodas (esteiras), com base da velocidade recebido pelo tópico inscrito.

Figura 107: sysCall_cleanup.

```
function sysCall_cleanup()
    if simROS then
        -- Shut down publisher and subscriber.
        -- Not really needed from a simulation script (automatic shutdown)
        simROS.shutdownSubscriber(sub)
    end
end
```

Fonte: O Autor (2022).

A **sysCall_cleanup()**, é executada uma vez antes da simulação terminar, restaurando a configuração inicial do objeto, cancelando a inscrição do retorno de chamada associado a esse assinante.

4 VALIDAÇÃO

Este capítulo visa validar a proposta DTMR, verificando os resultados obtidos pertinentes ao desenvolvimento deste projeto, embasado nos conceitos de Indústria 4.0, CPS, robótica móvel e DT.

4.1 VALIDAÇÃO DO DTMR TUPY

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 Coppelia-Sim.

4.1.1 Ambiente ROS

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.

4.1.1.1 Roscore

Este comando é 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 (Figura 108).

Figura 108: Iniciando Roscore no Terminal *desktop*.

```
roscore http://desktop:11311/
roscore http://desktop:11311/ 80x24
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://desktop:39267/
ros_comm version 1.14.13

SUMMARY
=====

PARAMETERS
* /rostdistro: melodic
* /rosversion: 1.14.13

NODES

auto-starting new master
process[roscout-1]: started with pid [4981]
ROS_MASTER_URI=http://desktop:11311/

setting /run_id to 3ca8676e-6866-11ed-867c-a86badd9a93
process[roscout-1]: started with pid [4995]
started core service [/roscout]
```

Fonte: O Autor (2022).

4.1.1.2 Arduino

Após iniciar o **roscore**, deve-se realizar o login no robô Tupy via **SSH**. Inicia-se o procedimento para configuração da 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. Então, os comandos da Figura 109 foram inseridos no terminal para realizar essa parametrização.

Figura 109: Configurando Rosserial-Arduino no Terminal.

```
tupy@tupy-robot: ~
tupy@tupy-robot: ~ 80x24
jean@desktop:~$ ssh tupy@192.168.15.22
tupy@192.168.15.22's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-1033-raspi2 armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

496 packages can be updated.
363 updates are security updates.

Last login: Tue Nov 15 21:04:27 2022 from 192.168.15.21
tupy@tupy-robot:~$ ls -l /dev/ttyACM*
crw-rw-rw- 1 root dialout 166, 0 nov 15 21:04 /dev/ttyACM0
tupy@tupy-robot:~$ sudo usermod -a -G dialout tupy
[sudo] password for tupy:
tupy@tupy-robot:~$ sudo chmod a+rw /dev/ttyACM0
tupy@tupy-robot:~$
```

Fonte: O Autor (2022).

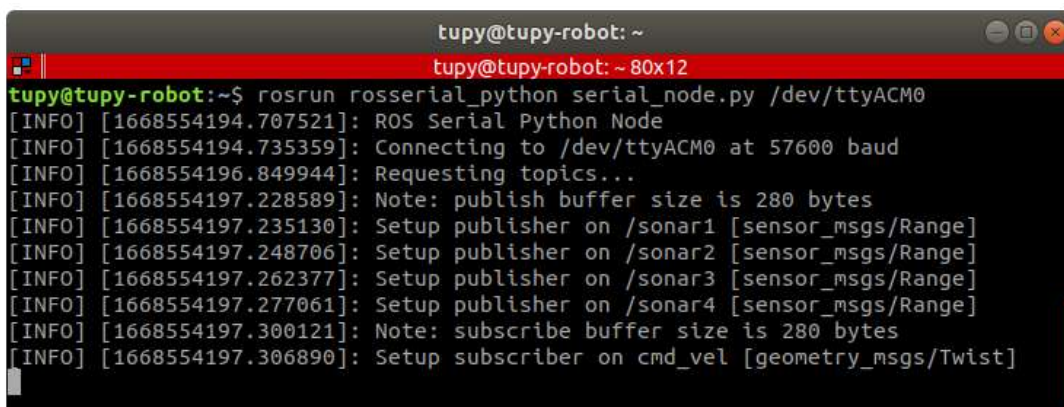
Após a conexão estabelecida do *desktop* com o robô Tupy, basta executar o comando:

- **export ROS_MASTER_URI=http://192.168.15.21:11311**
- **export ROS_IP=192.168.15.22**
- **roslaunch roserial_python serial_node.py /dev/ttyACM0**

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 *Internet Protocol* – Protocolo de Internet (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*, já o ROS IP pertence ao robô Tupy.

Em seguida, é possível notar que o comando **roslaunch** cria o **ROS Serial Python NODE**. Dessa forma, o ROS executa o *script* da conexão serial do Arduino na porta **tttyACM0**, publicando os Tópicos dos sonares (1, 2, 3 e 4) e inscrevendo as mensagens do **cmd_vel**, conforme Figura 110.

Figura 110: Iniciando Rosserial-Arduino no Terminal.

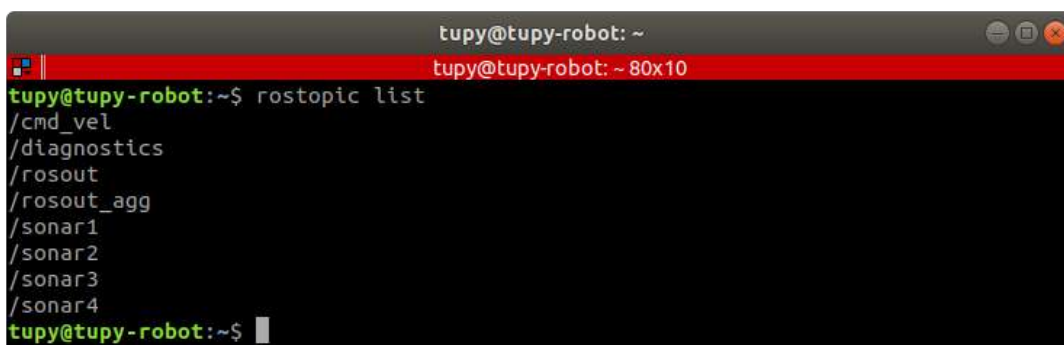


```
tupy@tupy-robot: ~  
tupy@tupy-robot: ~ 80x12  
tupy@tupy-robot:~$ roslaunch roserial_python serial_node.py /dev/ttyACM0  
[INFO] [1668554194.707521]: ROS Serial Python Node  
[INFO] [1668554194.735359]: Connecting to /dev/ttyACM0 at 57600 baud  
[INFO] [1668554196.849944]: Requesting topics...  
[INFO] [1668554197.228589]: Note: publish buffer size is 280 bytes  
[INFO] [1668554197.235130]: Setup publisher on /sonar1 [sensor_msgs/Range]  
[INFO] [1668554197.248706]: Setup publisher on /sonar2 [sensor_msgs/Range]  
[INFO] [1668554197.262377]: Setup publisher on /sonar3 [sensor_msgs/Range]  
[INFO] [1668554197.277061]: Setup publisher on /sonar4 [sensor_msgs/Range]  
[INFO] [1668554197.300121]: Note: subscribe buffer size is 280 bytes  
[INFO] [1668554197.306890]: Setup subscriber on cmd_vel [geometry_msgs/Twist]
```

Fonte: O Autor (2022).

Para verificar se o ROS está identificando os Tópicos ativos, o comando **rostopic list** pode ser digitado para verificar essa informação, conforme Figura 108.

Figura 111: Rostopic List.



```
tupy@tupy-robot: ~  
tupy@tupy-robot: ~ 80x10  
tupy@tupy-robot:~$ rostopic list  
/cmd_vel  
/diagnostics  
/rosout  
/rosout_agg  
/sonar1  
/sonar2  
/sonar3  
/sonar4  
tupy@tupy-robot:~$
```

Fonte: O Autor (2022).

A partir de então, utilizando a função **echo**, pode-se “ouvir” as informações que estão em cada tópico. Então, digitam-se os seguintes comandos em terminais separados:

- **rostopic echo /sonar1**
- **rostopic echo /sonar2**
- **rostopic echo /sonar3**
- **rostopic echo /sonar4**

Com isso, tem-se a visualização dos Tópicos (Figura 112) das distâncias lidas pelos sensores ultrassônicos, conforme programação do Arduino.

Figura 112: Echo 4 Sonares.

The image shows four terminal windows, each displaying the output of a different sonar sensor. Each window has a title bar indicating the user is 'tupy' on a 'tupy-robot' machine. The data is structured as follows:

Sonar	Seq	Stamp	Secs	Nsecs	Frame ID	Radiation Type	Field of View	Min Range	Max Range	Range
1	3659	1668554685	781625892	"/sonar1"	0	0.259	0.0	2.0	154.0	
2	569	1668554685	781625892	"/sonar2"	0	0.259	0.0	2.0	149.0	
3	467	1668554685	781625892	"/sonar3"	0	0.259	0.0	2.0	57.0	
4	277	1668554685	714625892	"/sonar4"	0	0.259	0.0	2.0	117.0	
1	3660	1668554685	846625892	"/sonar1"	0	0.259	0.0	2.0	154.0	
2	570	1668554685	846625892	"/sonar2"	0	0.259	0.0	2.0	151.0	
3	468	1668554685	846625892	"/sonar3"	0	0.259	0.0	2.0	57.0	
4	278	1668554685	781625892	"/sonar4"	0	0.259	0.0	2.0	124.0	

Fonte: O Autor (2022).

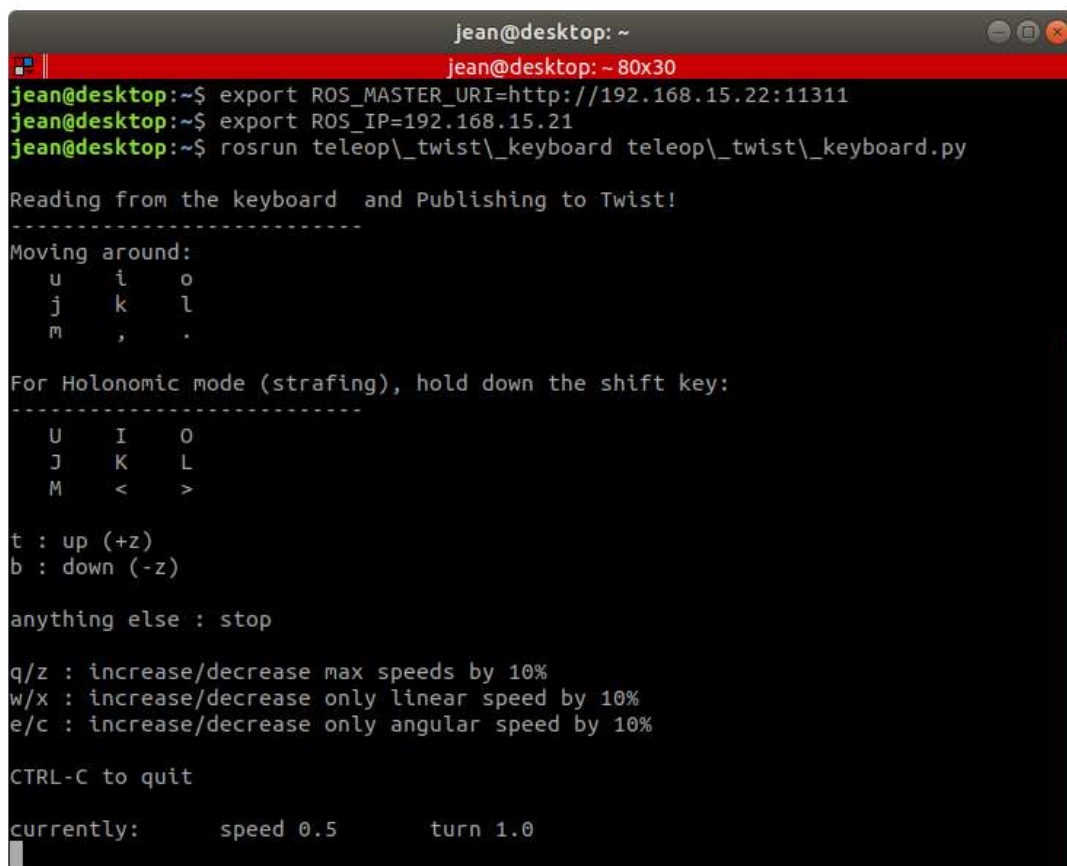
4.1.1.3 Teleoperação

Aqui será implementado o controle do robô físico e virtual simultaneamente com o uso do **cmd_vel**. Para iniciar a interface deve digitar-se as seguintes linhas de comando no terminal:

- **export ROS_MASTER_URI=http://192.168.15.21:11311**
- **export ROS_IP=192.168.15.22**
- **roslaunch teleop_twist_keyboard teleop_twist_keyboard.py**

Como as interfaces de inscrição deste foi iniciada anteriormente, o pacote então publica as mensagens vetoriais conforme a Figura 113.

Figura 113: Iniciando Teleoperação.



```

jean@desktop: ~
jean@desktop: ~ 80x30
jean@desktop:~$ export ROS_MASTER_URI=http://192.168.15.22:11311
jean@desktop:~$ export ROS_IP=192.168.15.21
jean@desktop:~$ roslaunch teleop_twist_keyboard teleop_twist_keyboard.py

Reading from the keyboard and Publishing to Twist!
-----
Moving around:
   u   i   o
   j   k   l
   m   ,   .

For Holonomic mode (strafing), hold down the shift key:
-----
   U   I   O
   J   K   L
   M   <   >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 0.5      turn 1.0

```

Fonte: O Autor (2022).

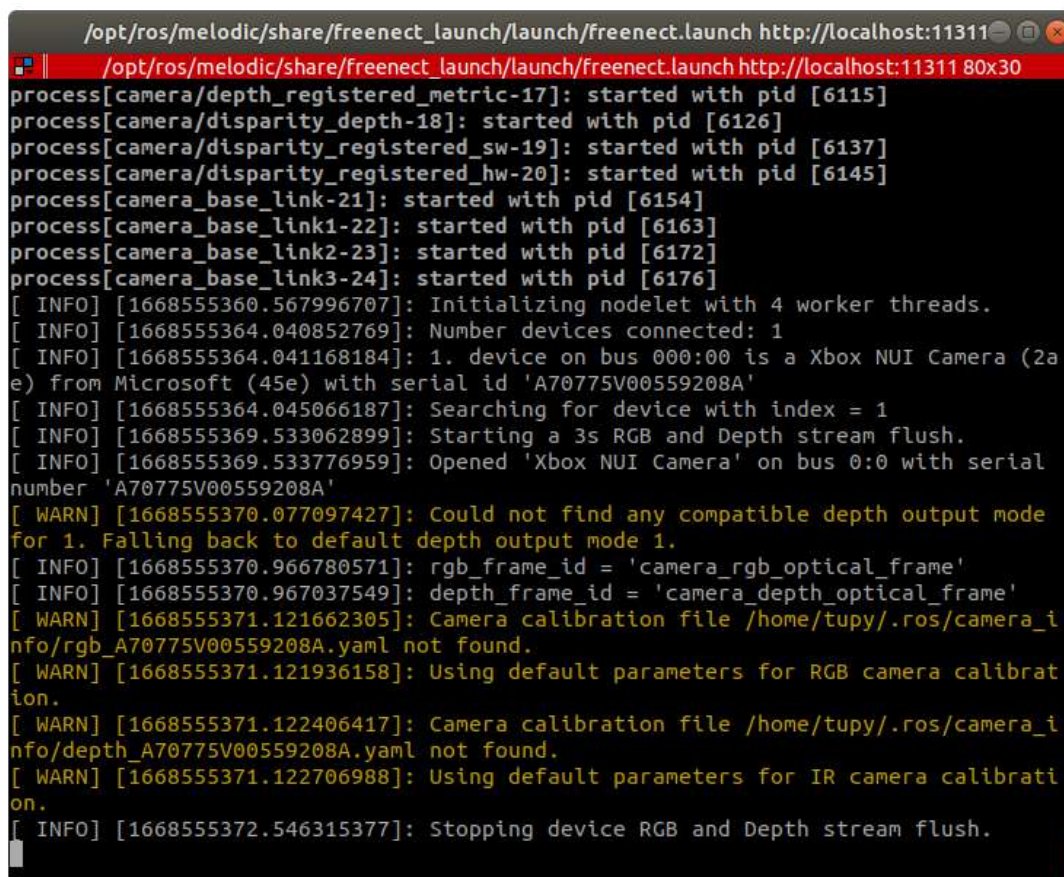
A partir de então, conforme a captura das teclas, o robô é movimentado com velocidades linear e angular definidas.

4.1.1.4 Kinect e OpenCV

Após a compilação dos drivers **freenect**, inicia-se com o **roslaunch** a publicação dos Tópicos da câmera do Kinect, conforme os seguintes comandos:

- **ROS_MASTER_URI=http://192.168.15.21:11311**
- **ROS_IP=192.168.15.22**
- **roslaunch freenect_launch freenect.launch**

Figura 114: Iniciando Pacote Freenect.



```

/opt/ros/melodic/share/freenect_launch/launch/freenect.launch http://localhost:11311
/opt/ros/melodic/share/freenect_launch/launch/freenect.launch http://localhost:11311 80x30
process[camera/depth_registered_metric-17]: started with pid [6115]
process[camera/disparity_depth-18]: started with pid [6126]
process[camera/disparity_registered_sw-19]: started with pid [6137]
process[camera/disparity_registered_hw-20]: started with pid [6145]
process[camera_base_link-21]: started with pid [6154]
process[camera_base_link1-22]: started with pid [6163]
process[camera_base_link2-23]: started with pid [6172]
process[camera_base_link3-24]: started with pid [6176]
[ INFO] [1668555360.567996707]: Initializing nodelet with 4 worker threads.
[ INFO] [1668555364.040852769]: Number devices connected: 1
[ INFO] [1668555364.041168184]: 1. device on bus 000:00 is a Xbox NUI Camera (2a
e) from Microsoft (45e) with serial id 'A70775V00559208A'
[ INFO] [1668555364.045066187]: Searching for device with index = 1
[ INFO] [1668555369.533062899]: Starting a 3s RGB and Depth stream flush.
[ INFO] [1668555369.533776959]: Opened 'Xbox NUI Camera' on bus 0:0 with serial
number 'A70775V00559208A'
[ WARN] [1668555370.077097427]: Could not find any compatible depth output mode
for 1. Falling back to default depth output mode 1.
[ INFO] [1668555370.966780571]: rgb_frame_id = 'camera_rgb_optical_frame'
[ INFO] [1668555370.967037549]: depth_frame_id = 'camera_depth_optical_frame'
[ WARN] [1668555371.121662305]: Camera calibration file /home/tupy/.ros/camera_i
nfo/rgb_A70775V00559208A.yaml not found.
[ WARN] [1668555371.121936158]: Using default parameters for RGB camera calibrat
ion.
[ WARN] [1668555371.122406417]: Camera calibration file /home/tupy/.ros/camera_i
nfo/depth_A70775V00559208A.yaml not found.
[ WARN] [1668555371.122706988]: Using default parameters for IR camera calibrati
on.
[ INFO] [1668555372.546315377]: Stopping device RGB and Depth stream flush.

```

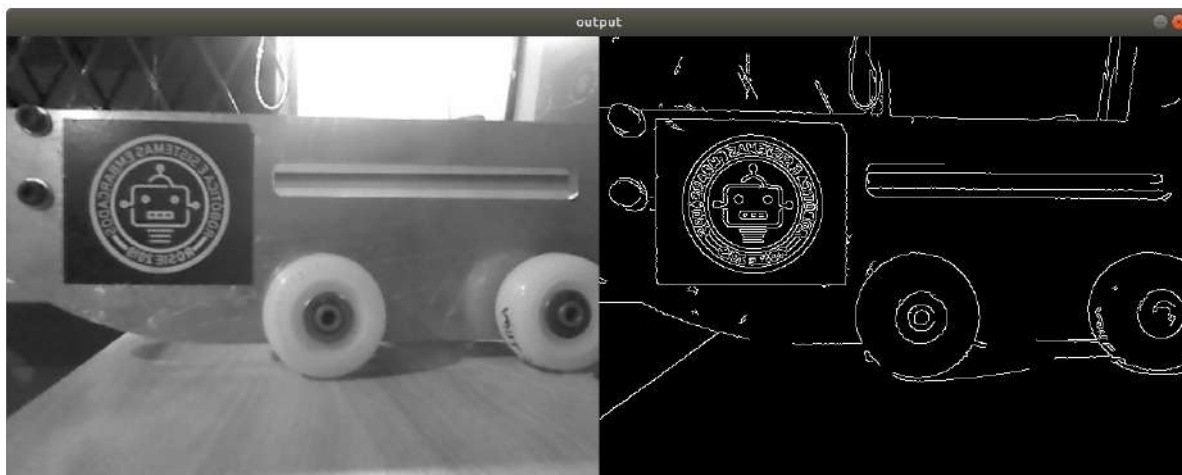
Fonte: O Autor (2022).

Nota-se, que os Tópicos estão prontos para serem inscritos. Então, inicia-se o pacote **Canny Kinect**, com os seguintes comandos no terminal:

- **export ROS_MASTER_URI=http://192.168.15.21:11311**
- **export ROS_IP=192.168.15.22**
- **roslaunch img_processor csf_robotCS_cannyKinect.py**

Com tudo certo, é retornado uma tela de visualização em tempo real com imagens em escala de cinza e com o rastreo das arestas, como se observa na Figura 115.

Figura 115: Canny Kinect em execução.



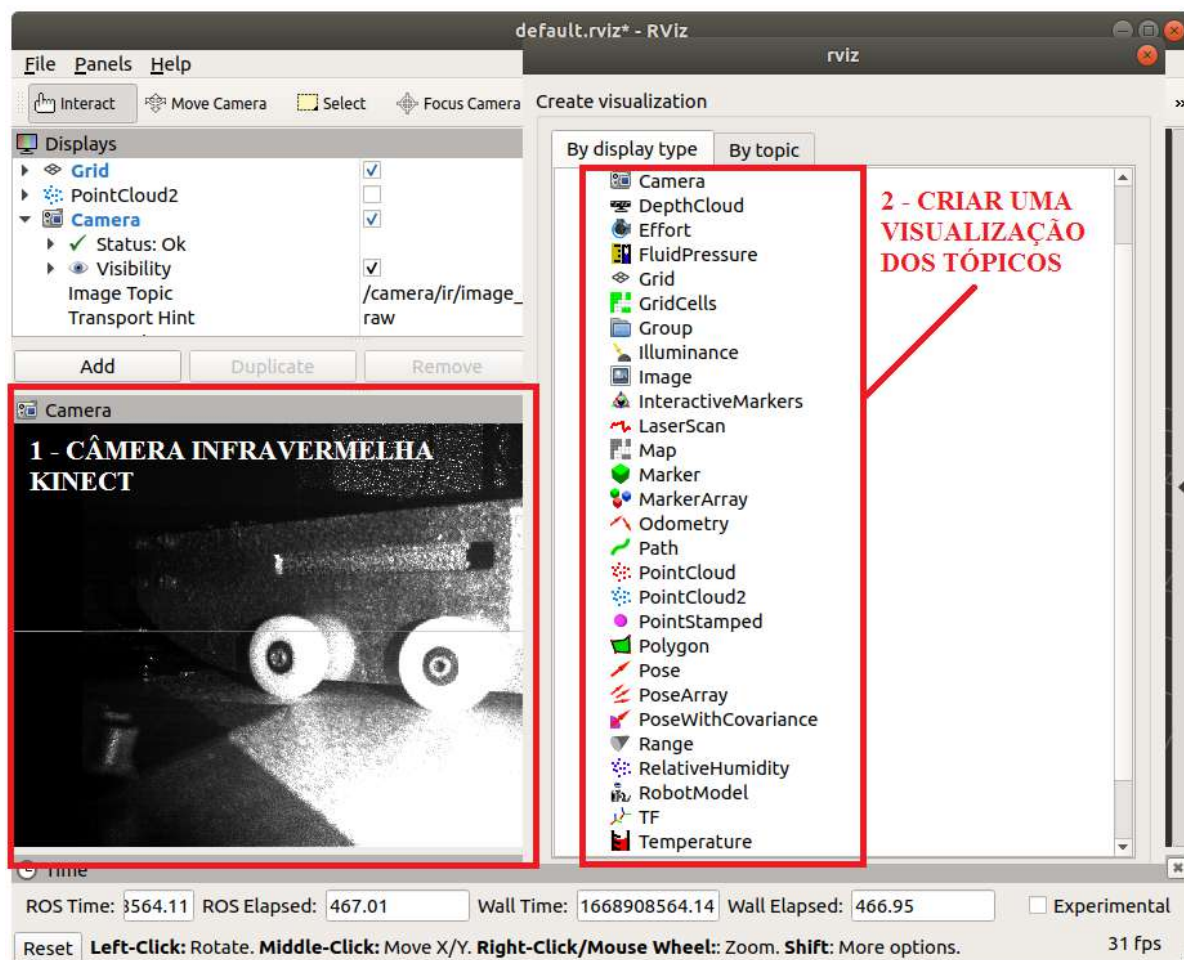
Fonte: O Autor (2022).

4.1.2 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 pode-se verificar que a comunicação entre os Nós do ROS está correta e se os drivers do Kinect foram iniciados corretamente. A Figura 116, demonstra através do detalhe (2) algumas destas opções disponíveis para criação. Já o detalhe (1), é possível visualizar a saída do tópico infravermelho do Kinect criado.

- **export ROS_MASTER_URI=http://192.168.15.21:11311**
- **export ROS_IP=192.168.15.22**
- **rviz**

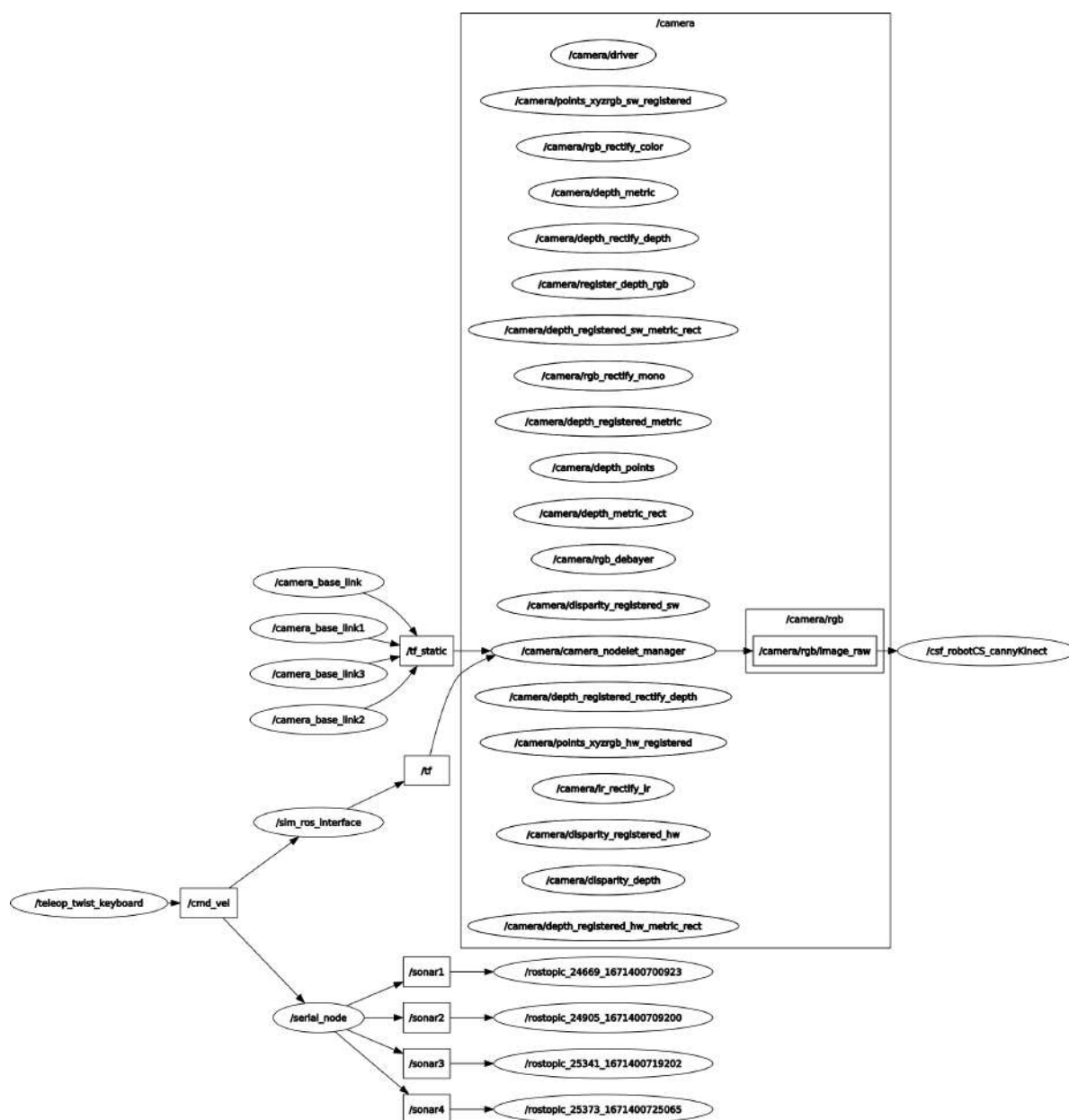
Figura 116: Visualizando o Tópico infravermelho da câmera.



Fonte: O Autor (2022).

Outra maneira de visualizar os tópicos ativos, é utilizar a ferramenta gráfica nativa do ROS `rtg_graph`, conforme Figura 117.

Figura 117: Arquitetura dos tópicos do robô Tupy com rtq_graph.



Fonte: O Autor (2022).

É possível notar (Figura 117) que o *driver* Freenect inicia vários Tópicos referentes a câmera. Porém, apenas o Tópico **/camera/rgb** é necessário para o funcionamento do pacote Canny. Já o **teleop_twist_keyboard** realiza o envio de 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 DT arquitetado pelo *framework* ROS.

4.1.3 Resumo da inicialização dos serviços do Gêmeo Digital

Portanto, em suma, o Tupy foi iniciado na sequência dos comandos inseridos no terminal do *desktop*, conforme ordem a seguir:

- No terminal 1, inicia-se o ROS *Master*:
 - jean@desktop: \$ roscore
- No terminal 2, inicia-se o *Freenect Driver*:
 - jean@desktop: \$ ssh tupy@192.168.15.22
 - tupy@tupy-robot: \$ export ROS_MASTER_URI=http://192.168.15.21:11311
 - tupy@tupy-robot: \$ export ROS_IP=192.168.15.22
 - tupy@tupy-robot: \$ roslaunch freenect_launch freenect.launch
- No terminal 3, inicia-se a comunicação serial do Arduino:
 - jean@desktop: \$ ssh tupy@192.168.15.22
 - tupy@tupy-robot: \$ export ROS_MASTER_URI=http://192.168.15.21:11311
 - tupy@tupy-robot: \$ export ROS_IP=192.168.15.22
 - tupy@tupy-robot: \$ rosrun rosserial_python serial_node.py /dev/ttyACM0
- No terminal 4, inicia-se a leitura do primeiro sensor ultrassônico:
 - jean@desktop: \$ export ROS_MASTER_URI=http://192.168.15.21:11311
 - jean@desktop: \$ export ROS_IP=192.168.15.21
 - jean@desktop: \$ rostopic echo /sonar1
- No terminal 5, inicia-se a leitura do segundo sensor ultrassônico:
 - jean@desktop: \$ export ROS_MASTER_URI=http://192.168.15.21:11311
 - jean@desktop: \$ export ROS_IP=192.168.15.21
 - jean@desktop: \$ rostopic echo /sonar2
- No terminal 6, inicia-se a leitura do terceiro sensor ultrassônico:
 - jean@desktop: \$ export ROS_MASTER_URI=http://192.168.15.21:11311
 - jean@desktop: \$ export ROS_IP=192.168.15.21
 - jean@desktop: \$ rostopic echo /sonar3
- No terminal 7, inicia-se a leitura do quarto sensor ultrassônico:

- jean@desktop: \$ export ROS_MASTER_URI=http://192.168.15.21:11311
- jean@desktop: \$ export ROS_IP=192.168.15.21
- jean@desktop: \$ rostopic echo /sonar4

- No terminal 8, inicia-se a execução do pacote Canny:

- jean@desktop: \$ export ROS_MASTER_URI=http://192.168.15.21:11311
- jean@desktop: \$ export ROS_IP=192.168.15.21
- jean@desktop: \$ rosrun img_processor csf_robotCS_cannyKinect.py

- No terminal 9, inicia-se a teleoperação com pacote Twist:

- jean@desktop: \$ export ROS_MASTER_URI=http://192.168.15.21:11311
- jean@desktop: \$ export ROS_IP=192.168.15.21
- jean@desktop: \$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py

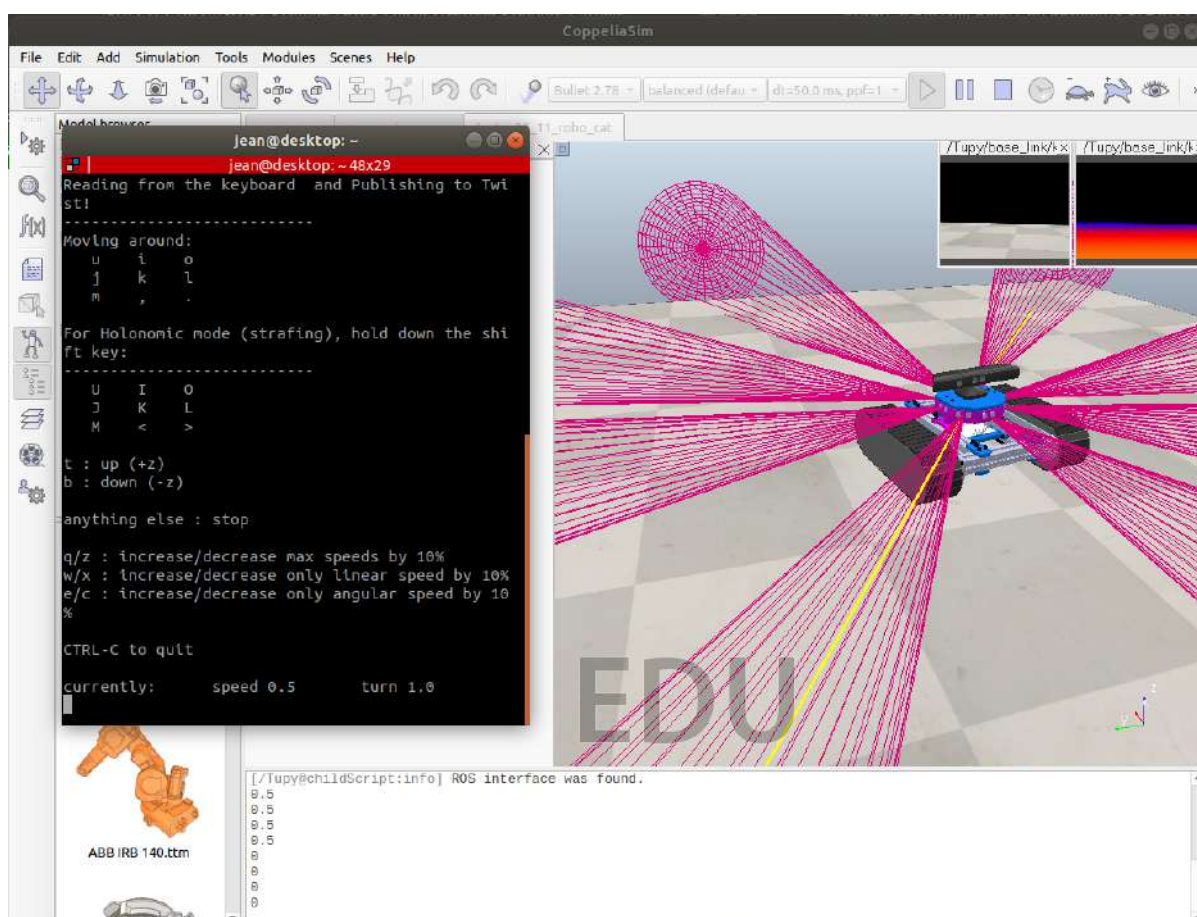
- No terminal 10, inicia-se o ambiente virtual do CoppeliaSim:

- jean@desktop: \$ export ROS_MASTER_URI=http://192.168.15.21:11311
- jean@desktop: \$ export ROS_IP=192.168.15.21
- jean@desktop: \$ coppelia

4.1.4 Tupy no ambiente ciber-físico como gêmeo digital

Finalizando todas as etapas demonstradas, parte-se para a inicialização do cenário virtual no CoppeliaSim, então abre-se o cenário com a modelagem do robô Tupy e inicia-se a simulação. Com isso, tem-se o DTMR Tupy no ambiente ciber, conforme Figura 118.

Figura 118: Tupy no ambiente CoppeliaSim.



Fonte: O Autor (2022).

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 119, do robô Tupy no ambiente real. Portanto, através deste desenvolvimento, tem-se o Gêmeo Digital, aplicados à robótica móvel DTMR.

Figura 119: Tupy no ambiente Real.

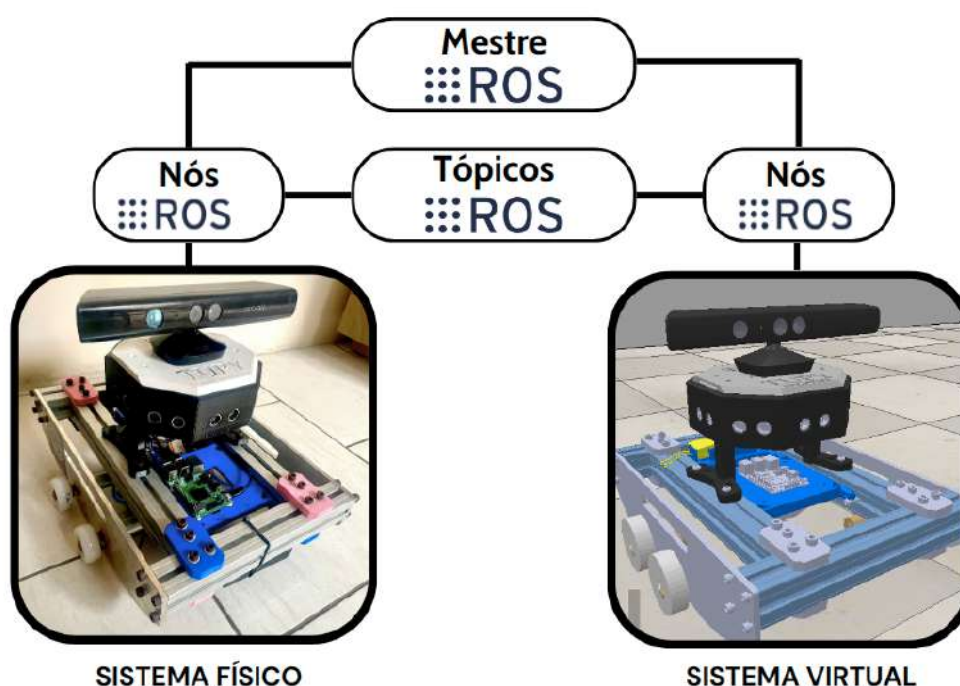


Fonte: O Autor (2022).

5 RESULTADOS E DISCUSSÃO

Neste capítulo, o objetivo é avaliar e discutir individualmente cada parte do sistema (virtual e físico), bem como a visão geral do sistema representado pela Figura 120. Assim, verificando se os resultados validados foram satisfatórios, as dificuldades apresentadas e os pontos positivos que enriqueceram o desenvolvimento deste trabalho.

Figura 120: Estrutura DTMR Tupy



Fonte: O Autor (2022).

5.1 DISCUSSÃO SOBRE OS RESULTADOS DO SISTEMA FÍSICO

Esta seção visa abordar os aspectos construtivos da montagem mecânica, eletrônica e computação do sistema físico do robô Tupy.

5.1.1 Mecânica

O resultado da montagem mecânica pode ser observado na Figura 121.

Figura 121: Resultado da montagem mecânica.



Fonte: O Autor (2022).

- Pontos positivos

Mecanicamente, teve-se grande êxito na execução deste trabalho, devido à utilização de uma plataforma pré-desenvolvida pelo grupo ROSIE. A versatilidade do perfil extrudado de alumínio, permitiu que ajustes pudessem ser feitos facilmente. Com isso, foi possível finalizar a integração no tempo estimado, para testes práticos com o robô Tupy.

A grata surpresa, foi a questão das rodas que, embora seu tamanho parecer pequeno para o porte do robô, se comportou adequadamente aos testes realizados em piso de cerâmica.

A questão da impressão 3D favoreceu a montagem dos componentes eletrônicos, permitindo o encaixe perfeito dos sensores ultrassônicos e fixação do Kinect. Os resultados das montagens podem ser verificadas nas Figuras 122 e 123.

Figura 122: Montagem mecânica com as peças desenvolvidas.



Fonte: O Autor (2022).

Figura 123: Montagem mecânica dos componentes eletrônicos.



Fonte: O Autor (2022).

- Pontos negativos

Um ponto negativo, foi o fato da base do suporte ter ficado deslocada (Figura 124 à esquerda), possivelmente devido a alguma incoerência nas medidas que ocorreu na parte da modelagem. O furo da tampa do octógono (Figura 124

à direita) não ficou ideal, precisaria estar com uma distância menor para facilitar a passagem do cabo com Núcleo de Ferrite do Kinect.

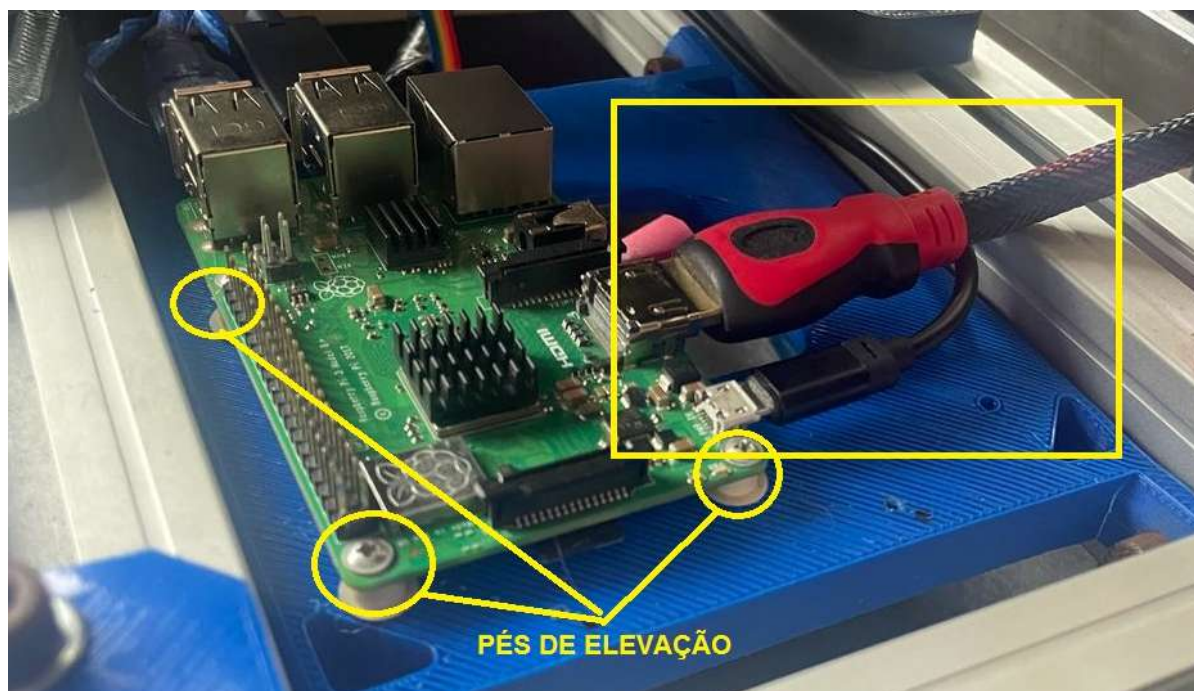
Figura 124: Pontos para melhoria



Fonte: O Autor (2022).

Já na montagem do RPi 3b+, como a peça suporte para placa não foi impressa, utilizou-se a que já estava no robô antigo. Desta forma, a montagem desta placa teve que ficar deslocada em relação ao centro, possibilitando a utilização das portas de interface HDMI e micro USB sem risco de quebra da placa. Também, foi necessário elevar a placa por três motivos: melhorar a troca térmica com o ambiente, devido aquecimento do processador ao iniciar os tópicos do ROS; e facilitar o encaixe dos laterais; facilitar o acesso do cartão micro SD (Figura 125).

Figura 125: Problemas Placa RPi



Fonte: O Autor (2022).

5.1.2 Eletrônica

O resultado do desenvolvimento eletrônico pode ser verificado no desenho esquemático elaborado no *software* Proteus, presente no Apêndice A.

- Pontos positivos

A grande vantagem no desenvolvendo desta integração com a utilização de plataformas prontas como Raspberry Pi, Arduino Uno e Ponte H (L298N), foi o fato de permitir a integração destes sistemas com o *framework* ROS. Desta forma, foi possível utilizar as conexões via USB e Wi-Fi para realizar a interface entre Arduino, RPi e *desktop*, minimizando as ligações elétricas entre os periféricos.

Outro ponto interessante, foi a utilização do Kinect. Este é um periférico conta vários componentes integrados, como câmera RGB, sensor infravermelho, sensor de profundidade, microfones e motor para angulação vertical do sensor. Desta forma, o uso desse dispositivo gera diversas oportunidades para interação com o ambiente.

- Pontos negativos

Infelizmente, não foi possível finalizar o circuito de alimentação com a bateria há tempo. Dessa forma, não se pôde validar a autonomia de carga da bateria

do robô e verificar a distância máxima de teleoperação. Entretanto, realizaram-se todos os testes principais relacionados ao DTMR, com o robô ligado pelas fontes de alimentação ligadas à tomada.

Outra questão é com relação aos sonares, já que, inicialmente, não se tinha posse de todos os sensores (oito) para realização dos testes. Desta forma, não se realizou a multiplexação, com isso, foi utilizado 8 I/Os do microcontrolador ATmega328p para a ligação dos quatro sensores. Isso acabou acarretando dificuldades para a elaboração de um controle dos motores via PWM, devido à falta de saídas que esta configuração gerou.

5.1.3 Computação

A computação foi crucial no desenvolvimento desta integração, pois o controle do DTMR, baseou-se, principalmente, na questão computacional. Portanto, houve ampla exploração deste tópico.

- Pontos positivos

O grande ponto positivo, foi desenvolvimento pessoal neste aspecto, devido o contato com diversas ferramentas computacionais como SolidWorks, Proteus, Arduino IDE. Já em conceitos da Tecnologia da Informação (TI) como redes, instalação e configuração de sistemas operacionais, gerenciamento de arquivos, entre outros, foi possível verificar o quão integrado sistemas robóticos estão relacionados com a informática. Por fim, linguagens de programação com o desenvolvimento da programação em Python, C/C++ e contato com a linguagem LUA, utilizada no CoppeliaSim.

- Pontos negativos

Inicialmente houve um pouco de dificuldade, pela curva de aprendizagem, em relação aos Sistemas Operacionais Linux e, principalmente, ao ROS, simulador robótico CoppeliaSim. Entender desses sistemas levou-se um certo tempo para ser possível realizar a integração de todo esse ambiente digital.

No começo foi tentado realizar a instalação do SO Linux Ubuntu em máquina virtual com o *software* VirtualBox, porém a instalação do sistema apresentou diversos problemas relacionados aos *drivers* USBs do Kinect e Arduino, impossibilitando prosseguimento do projeto por esta via. Dessa forma, foi necessário realocar o

disco local físico do *desktop* para realizar a instalação do Linux lado a lado (*dual-boot*) com o Windows.

Também, houve dificuldades com o RPi, por conta de ser o primeiro contato com esta plataforma. A localização de um sistema adequado para o processador (armhf) que suportasse a versão correta do ROS, no caso Melodic, demandou várias tentativas de instalação do SO.

Dificuldades foram encontradas no *framework* ROS, principalmente em integrar a câmera Kinect, por conta da instalação manual *drivers*, devido à versão pré-compilada estar defasada. Inicialmente, não se foi atentado para a correta execução do ROS MASTER, com isso o *framework* não conseguia encontrar os Nós no RPi e/ou *desktop*.

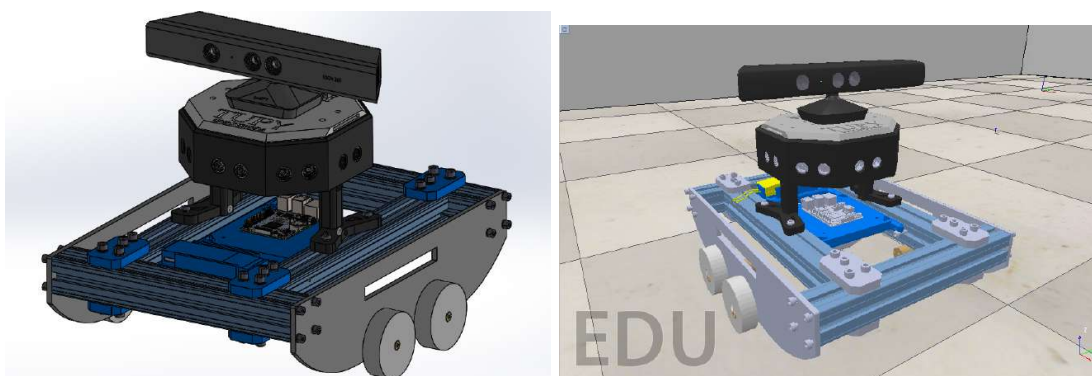
5.2 DISCUSSÃO SOBRE OS RESULTADOS DO SISTEMA VIRTUAL

Esta seção visa abordar os aspectos construtivos da mecânica, eletrônica e computação do sistema virtual do robô Tupy.

5.2.1 Mecânica

Os resultados da virtualização da plataforma mecânica do DTMR Tupy podem ser vistos na Figura 126, sendo a esquerda o modelo em 3D no SolidWorks, e a direita o modelo 3D no CoppeliaSim.

Figura 126: Resultado da modelagem do robô Tupy.



Fonte: O autor (2022).

- Pontos positivos

Na modelagem obteve-se êxito na combinação da parte inferior (estrutura de chassi e rodas), feito anteriormente em CATIA, com as peças atualizadas feitas

no SolidWorks. Comprova-se outra vantagem de se utilizar o SolidWorks – sua grande disponibilidade de componentes já modelados para o *software* realizados pela comunidade — assim os que foram incluídos no projeto já prontos a partir de importação são: as plataformas RPi e Arduino UNO R3, e o Kinect.

- Pontos negativos

O modelo STEP exportado para o SolidWorks não permitiu alterações em seu esboço original, dificultando o processo de modificação dos componentes da estrutura. Dessa forma, necessitou-se remodelar as peças do zero, como a base para a plataforma RPi, a base do Kinect e seus componentes de sustentação.

A ferramenta de exportação do SolidWorks **SW2URDF**, não permitiu exportar todas as texturas do robô em sua construção total, assim exportou-se cada peça separadamente, para reconstruí-las no CoppeliaSim e então obter o robô Tupy completo.

Outro ponto, foi a questão da movimentação do robô Tupy com quatro rodas no ambiente virtual. Onde não houve a obtenção de resultados satisfatórios para esta locomoção. Dessa forma, as rodas foram substituídas por esteiras, garantindo fluidez na exploração do ambiente digital e, também, deixando a modelagem pronta para a modificação das rodas por esteiras do robô Tupy físico.

5.2.2 Eletrônica

Os resultados do sistema eletrônico virtual, foram suficientes para a demonstração prática da integração dos sistemas ciber-físicos do DTMR. Entretanto, algumas questões poderiam ser melhores tratadas, dedicando mais tempo para o desenvolvimento da programação eletrônica.

- Pontos positivos

O controle dos motores foi facilitado pelo pacote **ros-melodic-teleop-twist-keyboard** do ROS. Essa questão foi essencial para a movimentação do DTMR, pois o sinal era enviado simultaneamente para o robô físico e virtual. Com isso, pôde-se efetuar a teleoperação diretamente via teclado, sem a necessidade de uma programação complexa.

- Pontos negativos

Os sensores ultrassônicos apresentaram pequenas variações ao realizar algumas leituras de distância via terminal Linux, gerando também um pequeno atraso na resposta do sinal. Um ponto que ficou pendente foi a inclusão das leituras dos sensores no ambiente virtual do simulador CoppeliaSim.

A resposta da movimentação do robô virtual no CoppeliaSim não ficou tão adequada, devido ao *feedback* das rodas físicas. Com isso, não foi possível sincronizar a movimentação física com a virtual, pois não se sabia o quanto o robô havia se deslocado fisicamente. Esse problema, poderia ser contornado caso com a inserção de um *encoder* nas rodas.

5.2.3 Computação

Os resultados da virtualização do sistema do DTMR Tupy foram satisfatórios, pois se obteve êxito na integração das plataformas RPi, Arduino UNO R3 e *desktop* com o Sistema Operacional Robótico ROS.

- Pontos positivos

Através da utilização do ROS, o processo de programação da virtualização com o sistema Linux, ficou simplificada. Este sistema dedicado, permitiu que fosse possível arquitetar o DTMR, apenas iniciando Nós dedicados para esta finalidade. A utilização de bibliotecas desenvolvidas pela comunidade, também favoreceu este aspecto.

A programação dos sensores e atuadores ficou otimizada através do *software* de interface dedicada (Arduino IDE). Já a comunicação serial do Arduino com o Raspberry Pi e *desktop*, foi proporcionada pelo pacote **rosserial-arduino** que promoveu a integração dos periféricos de baixo nível com sistemas computacionais de alto nível.

Um ponto interessante foi o conhecimento obtido através do uso do Kinect. No primeiro momento, não se tinha noção que um componente dedicado para *videogames* pudesse ser amplamente explorado pela comunidade através do *driver* **Freenect**.

A simulação foi um ponto chave, pois com ela permitiu que todo o desenvolvimento do projeto pudesse ser feito. A ferramenta CoppeliaSim abre um imenso leque para testes com o robô e, principalmente, a possibilidade da criação de um DTMR.

- Pontos negativos

Na virtualização do sistema operacional robótico, foram encontrados problemas relacionados a programação do Kinect e sonares. Infelizmente, não foi possível realizar a leitura destes sensores em tempo real dentro do ambiente virtual do CoppeliaSim. A ideia seria receber esses dados no cenário digital, sem a necessidade de terminais extras como Canny Kinect e Sonares (1 ao 4), para a visualização destas informações.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

6.1 CONSIDERAÇÕES FINAIS E CONCLUSÃO.

Este trabalho apresentou uma proposta da criação de um gêmeo digital em robótica móvel como ferramenta de aprendizagem em CPS. O trabalho realizou seu objetivo principal de virtualização, ao criar um DTMR através do *framework* ROS, gerando assim infinitas possibilidades de controle e percepção.

Dessa maneira, o conjunto de pesquisa desenvolvido ao fim deste trabalho é de grande importância acadêmica e técnica, visto a utilização de conhecimentos específicos abordados ao longo do curso de graduação de Engenharia de controle e automação. A criação de um DTMR contribuiu fortemente nesse sentido, através do uso de simuladores robóticos, (CoppeliaSim), arquitetura de sistemas operacionais computacionais (Linux) e robóticos (ROS), uso de plataformas de desenvolvimento como Arduino com microcontrolador (ATmega328p) e Raspberry Pi 3b+ com microprocessador (Cortex-A53 (ARMv8) 64-bits), ferramentas CAD/CAE como SolidWorks e Proteus, aplicação de filtros digitais e processamento de imagens.

Ademais, os robôs móveis quando desenvolvidos em projetos acadêmicos possibilitam o uso de ferramentas que se enquadram na robótica educacional, e na educação inclusiva, pois fornecem uma aplicação com alta tecnologia acessível com um custo relativamente baixo.

Este projeto tem também tem o objetivo de ser continuado, e aprimorado pelos estudantes de engenharia. Em vista disso, promove o compartilhamento de todo o material desenvolvido nesta pesquisa para acesso da comunidade acadêmica. Disponível em: (https://github.com/rosie-projects/dtmr_tupy).

6.2 PROPOSTAS PARA TRABALHOS FUTUROS

Com a coleta de dados pelos testes e resultados aqui apresentados, pode-se colocar diversos pontos para melhorias no projeto, agregando a outras pesquisas e desenvolvimentos futuros.

- Em relação ao sistema de locomoção do robô, propõe-se o uso de um sistema por esteiras, para aumentar a versatilidade de ambientes em que poderá se locomover, e aproveita o tipo de estrutura que o robô foi inicialmente projetado.
- Em relação à posição do robô no espaço, propõe-se adicionar sensor a LiDAR, para uma maior precisão e alcance nas medições do perímetro, e uma *Inertial*

Measurement Units – Unidade de Medição Inercial (IMU) para mensurar a posição absoluta do robô no ambiente.

- Em relação a uma movimentação do robô virtualmente, propõe-se a adição de um kit para técnica de odometria, capaz de fornecer a medida de deslocamento total do robô; de um *encoder* para aferição de velocidade; de controle da velocidade dos motores por PWM; e do uso de um controle analógico para teleoperação do robô.
- Em relação à simulação do robô, a implementação de um pacote *Simultaneous Localization and Mapping* – Localização e Mapeamento Simultâneos (SLAM) (Simultaneous Localization and Mapping) para ROS, a fim de criar um mapa do ambiente enquanto o robô se movimenta.
- Em relação à percepção do ambiente físico no ambiente virtual, implementar a visualização da imagem do Kinect em tempo real e leitura dos sensores ultrassônicos, no simulador CoppeliaSim.
- Em relação aos oito sensores ultrassônico, o desenvolvimento de um algoritmo que realize a multiplexação dos oito sensores ultrassônicos, para otimização das GPIOs do microcontrolador.
- Em relação ao fornecimento de energia do robô, preparar um circuito específico para inclusão de uma bateria.
- Em relação ao cenário, realizar a modelagem do campus Unicuritiba (Milton Viana) no ambiente virtual do *software* CoppeliaSim.
- Em relação ao sistema operacional integrador dos sistemas físicos e virtuais, a implementação do pacote adicional ROS2, por buscar adequação as atualizações fornecidas pelo desenvolvedor, e pelos novos recursos disponíveis.

REFERÊNCIAS

- ADAFRUIT. *mBot Robot Kit - Bluetooth Version - by Makeblock*. 2022. Url <https://www.adafruit.com/product/3640>.
- AL TAHTAWI, A. Kalman filter algorithm design for hc-sr04 ultrasonic sensor data acquisition system. *IJITEE (International Journal of Information Technology and Electrical Engineering)*, v. 2, 07 2018.
- ALGHAMDI, S. et al. Additive manufacturing of polymer materials: Progress, promise and challenges. *Polymers*, v. 13, 02 2021.
- ALIMISIS, D. Educational robotics: Open questions and new challenges. *Themes in Science and Technology Education*, v. 6, p. 63–64, 01 2013.
- ALNAHAM, M.; SULIMAN, M. Microcontroller-based system for voltage monitoring, protection and recovery using proteus vsm software. *International Journal of Computer Applications*, v. 118, p. 1–5, 05 2015.
- ANGEL-FERNANDEZ, J.; VINCZE, M. Towards a formal definition of educational robotics. In: . [S.l.: s.n.], 2018.
- ANUMBA, C.; AKANMU, A.; MESSNER, J. Towards a cyber-physical systems approach to construction. In: . [S.l.: s.n.], 2010. p. 528–538. ISBN 978-0-7844-1109-4.
- ARCBOTICS. *Sparki – Programmable Arduino STEM Robot Kit for Kids*. 2022. Url <http://arcbotics.com/products/sparki/>.
- ARMENDIA, M. et al. *TwinControl- A Digital Twin Approach to Improve Machine Tools Lifecycle*. 1 ed. [S.l.]: Springer, 2019. ISBN 978-3-030-02202-0.
- ARUP, D. et al. A review on filament materials for fused filament fabrication. *Journal of Manufacturing and Materials Processing*, v. 5, n. 3, 2021. ISSN 2504-4494. Disponível em: <<https://www.mdpi.com/2504-4494/5/3/69>>.
- ASCENCIO, A. F. G.; CAMPOS, E. A. V. de. *Fundamentos da programação de computadores*. [S.l.]: Pearson Educación, 2008.
- AVÉROUS, L. Polylactic acid: Synthesis, properties and applications. In: . [S.l.: s.n.], 2008.
- AXELSON, J. *USB Complete: The Developer's Guide*. Lakeview Research, 2009. (Complete Guides Series). ISBN 9781931448086. Disponível em: <<https://books.google.com.br/books?id=YapRPQAACAAJ>>.
- BAHETI, R. S.; GILL, H. Cyber-physical systems. *2019 IEEE International Conference on Mechatronics (ICM)*, 2019.
- BANDYOPADHYAY, A.; BOSER, S. *Additive Manufacturing*. 2 ed. [S.l.]: Boca Raton, 2019. ISBN 9780429466236.
- BEN-ARI, M.; MONDADA, F. Elements of robotics. *Elements of Robotics*, 01 2018.

BONGARD, J. Probabilistic robotics. sebastian thrun, wolfram burgard and dieter fox. (2005, mit press.) 647 pages. *Artificial Life*, v. 14, n. 2, p. 227–229, 2008.

BRAUNL, T. *Embedded robotics : from mobile robots to autonomous vehicles with Raspberry Pi and Arduino*. Fourth edition. Singapore: Springer, 2022. ISBN 9789811608049.

BRENKEN, B. et al. Fused filament fabrication of fiber-reinforced polymers: A review. *Additive manufacturing*, v. 21, p. 1–16, 2018.

BRUZZONE, L.; QUAGLIA, G. Review article: locomotion systems for ground mobile robots in unstructured environments. *Mechanical Sciences*, v. 3, n. 2, p. 50, 2012. Disponível em: <<https://ms.copernicus.org/articles/3/49/2012/>>.

CARVALHO, N.; CAZARINI, E. Industry 4.0 - what is it? In: ORTIZ, J. H. (Ed.). *Industry 4.0 - Current Status and Future Trends*. 1 ed. <https://www.intechopen.com/books/9426>: intechopen, 2020. v. 1, p. 3–11.

CASTANON, G. A. O que é o construtivismo. *Cadernos de História e Filosofia da Ciência*, v. 1, n. 2, p. 209–242, 2015.

CELES, W.; FIGUEIREDO, L.; IERUSALIMSKY, R. *A Linguagem Lua e suas Aplicações em Jogos*. [S.l.: s.n.], 2004.

CHEN, A. et al. Soft robotics: Definition and research issues. In: . [S.l.: s.n.], 2017. p. 366–370.

COPPELIA. *CoppeliaSim User Manual*. 2022. <https://www.coppeliarobotics.com/helpFiles/>.

DALCÍN, L.; PAZ, R.; STORTI, M. Mpi for python. *Journal of Parallel and Distributed Computing*, v. 65, n. 9, p. 1108–1115, 2005. ISSN 0743-7315. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0743731505000560>>.

DANIELA, L.; LYTRAS, M. Educational robotics for inclusive education. *Technology, Knowledge and Learning*, v. 24, 12 2018.

DAWSON, M. *Python Programming for the Absolute Beginner*. 1 ed. [S.l.]: NovaTech, 2003. ISBN 1592000738.

DEDA, D.; MAGRIN, C. E. Robô móvel microcontrolado como uma ferramenta de ensino. *REVISTA DE EXTENSÃO E INICIAÇÃO CIENTÍFICA DA UNISOCIESC*, v. 4, n. 1, p. 64–65, nov. 2020. Disponível em: <<https://reis.unisociesc.com.br/index.php/reis/article/view/54>>.

DENARDIN, G.; BARRIQUELLO, C. *Sistemas operacionais de tempo real e sua aplicação em sistemas embarcados*. Blucher, 2019. ISBN 9788521213970. Disponível em: <<https://books.google.com.br/books?id=FrqxDwAAQBAJ>>.

DIKHANBAYEVA, D. et al. Assessment of industry 4.0 maturity models by design principles. *Sustainability*, v. 12, 11 2020.

- DOCS.ROS. *Understanding topics*. 6 2022.
 Url<https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html>.
- DRATH, R.; HORCH, A. Industrie 4.0: Hit or hype? [industry forum]. *Industrial Electronics Magazine, IEEE*, v. 8, p. 56–58, 06 2014.
- EL-LAITHY, R. A.; HUANG, J.; YEH, M. Study on the use of microsoft kinect for robotics applications. In: *Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium*. [S.l.: s.n.], 2012. p. 1280–1288.
- ESTERLE, L.; GROSU, R. Cyber-physical systems: challenge of the 21st century. *e and i Elektrotechnik und Informationstechnik*, v. 133, p. 1–5, 11 2016.
- ESTORILIO, C.; HATAKEYAMA, K. Sistemas cad/cae/cam integrados aperfeiçoam a injeção de plásticos. *Plástico Industrial*, v. 3, p. 18, 03 1998.
- FALCÃO, A. C. R. d. A. Indústria 4.0: uma análise utilizando revisão bibliográfica sistemática. *Industrial Electronics Magazine, IEEE*, 2019.
- FIGUEIRA, R. J. C. D. M. *CAD/CAE/CAM/CIM*. 1-123 p. Monografia (Licenciatura) — INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO, Portugal, 2002/2003.
- FLORES, P.; CLARO, J. C. P. *Cinemática de mecanismos 1: introdução ao estudo de mecanismos*. 2005.
- FOUNDATION, P. S. *General Python FAQ*. 2022.
 Url<https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place>.
- FOUNDATION, R. P. *Learn to program in Python*. 2022. Url
<https://www.raspberrypi.org/courses/learn-python>.
- GERACI, A. et al. Ieee standard computer dictionary: A compilation of ieee standard computer glossaries. *IEEE Std 610*, p. 1–217, 1991.
- GIBSON, I.; ROSEN, D.; STUCKER, B. *Additive Manufacturing Technologies 3D Printing, Rapid Prototyping and Direct Digital Manufacturing*. 2 ed. [S.l.]: Springer, 2015.
- GOTRO, J. *Characterization of Thermosets Part 21: Tensile Testing of Polymers; A Molecular Interpretation*. 2017. Url
<https://polymerinnovationblog.com/characterization-thermosets-part-21-tensile-testing-polymers-molecular-interpretation/>.
- GUIRADO, R. R. S.; VACAS, G. G.; ALABANDA, O. R. Teaching cad/cam/cae tools with project-based learning in virtual distance education. *Education and Information Technologies*, v. 27, p. 5051 – 5073, 2022.
- HABIB, M. K.; CHIMSOM, C. Industry 4.0: Sustainability and design principles. In: *2019 20th International Conference on Research and Education in Mechatronics (REM)*. [S.l.: s.n.], 2019. p. 1–8.

- HAMAD, K. et al. Properties and medical applications of polylactic acid: A review. *Express Polymer Letters*, v. 9, p. 435–455, 2015.
- HENNING, K. Recommendations for implementing the strategic initiative industrie 4.0. In: . [S.l.: s.n.], 2013.
- HERMANN, M.; PENTEK, T.; OTTO, B. Design principles for industrie 4.0 scenarios. *Industrial Electronics Magazine, IEEE*, 2016.
- HORTMAN, M. *INTRODUCTION TO SOLIDWORKS*. 2013. <<https://openwa.pressbooks.pub/testmhrtc/chapter/chapter-1-2/>>.
- HU, F. *Cyber-Physical Systems: Integrated Computing and Engineering Design*. [S.l.: s.n.], 2014. 3 p. ISBN ISBN 9781466577008.
- HUMCKE, D. Interactive computer graphics in c.a.d. *Computer-Aided Design*, v. 8, n. 4, p. 227–232, 1976. ISSN 0010-4485. Disponível em: <<https://www.sciencedirect.com/science/article/pii/0010448576901561>>.
- IEEE-802.11. Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, p. 1–4379, 2021.
- IEEE-802.15. Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, p. 1–800, 2020.
- IERUSALIMSCHY, R. *Uma Introdução à Programação em Lua*. 2022. Url <http://www.inf.puc-rio.br/noemi/sd-13/introlua.pdf>.
- IERUSALIMSCHY, R.; FIGUEIREDO, L. H. de; CELES, W. The evolution of lua. In: . New York, NY, USA: Association for Computing Machinery, 2007. ISBN 9781595937667. Disponível em: <<https://doi.org/10.1145/1238844.1238846>>.
- ILANKOVIĆ, N. et al. Smart factories - the product of industry 4.0. In: . [S.l.: s.n.], 2020. v. 7, p. 19–30.
- JAZDI, N. Cyber physical systems in the context of industry 4.0. In: *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*. [S.l.: s.n.], 2014. p. 1–4.
- KAGERMANN, H. et al. *Industrie 4.0 in a Global Context: Strategies for Cooperating with International Partners (acatech STUDY)*. [S.l.: s.n.], 2016. ISBN ISSN 2192-6174.
- KANEHIRO, F. et al. Open architecture humanoid robotics platform. In: . [S.l.: s.n.], 2002. v. 21, p. 24 – 30 vol.1. ISBN 0-7803-7272-7.
- KOENIG, N.; HOWARD, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: . [S.l.: s.n.], 2004. p. 2149 – 2154 vol.3. ISBN 0-7803-8463-6.

LABCENTER. *About Labcenter*. 2022. Url <https://www.labcenter.com/about/>.

LEE, E. A.; SESHIA, S. A. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. [S.l.: s.n.], 2016. 5 p. ISBN ISBN 9780262533812.

LEE, K. *Principles of CAD/CAM/CAE Systems*. USA: Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN 0201380366.

LEGO. *Lego Mindstorms Education EV3*. 2022. Url <https://education.lego.com/en-us/products/lego-mindstorms-education-ev3-core-set/5003400#lego-mindstorms-education-ev3>.

LIBERTY, J. *C++ de A a Z*. 1 ed. [S.l.]: Campus, 1999.

LINUX. *What Is Linux*. 2022. Url <https://www.linux.com/what-is-linux>.

LUA. *A Linguagem de Programação LUA*. 2022. Url <https://www.lua.org/portugues.html>.

MAGRIN, C. et al. Promovendo a aprendizagem da robótica nas escolas com metodologias ativas e o desenvolvimento de um robô móvel acessível para redução das desigualdades sociais. *Anais do Computer on the Beach*, 2022.

MAGRIN, C. E.; BRITO, R. C.; TODT, E. A systematic mapping study on multi-sensor fusion in wheeled mobile robot self-localization. In: *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*. [S.l.: s.n.], 2019. p. 132–137.

MAGRIN, C. E.; DEL CONTE, G.; TODT, E. Creating a digital twin as an open source learning tool for mobile robotics. In: *2021 Latin American Robotics Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR) and 2021 Workshop on Robotics in Education (WRE)*. [S.l.: s.n.], 2021. p. 13–18.

MANZANO, J. A. N. G. *Introdução à linguagem Python*. 1 ed. [S.l.]: NovaTech, 2018. ISBN 978-85-7522-714-5.

MARTIN, O.; AVÉROUS, L. Poly(lactic acid): plasticization and properties of biodegradable multiphase systems. *Polymer*, v. 42, p. 6209–6219, 2001.

MARTINEZ, A.; FERNNDEZ, E. *Learning ROS for Robotics Programming*. [S.l.]: Packt Publishing, 2013. ISBN 1782161449.

MARTINEZ, K. et al. Thermoplastic starch (tps)/polylactic acid (pla) blending methodologies: A review. *Journal of Polymers and the Environment*, v. 30, 01 2022.

MATARIĆ, M. J. *The Robotics Primer*. 1 ed. [S.l.]: Massachusetts Institute of Technology, 2007. ISBN 978-0-262-63354-3.

MATSAS, E.; VOSNIAKOS, G.-C.; BATRAS, D. Prototyping proactive and adaptive techniques for human-robot collaboration in manufacturing using virtual reality. *Robot. Comput.-Integr. Manuf.*, Pergamon Press, Inc., USA, v. 50, n. C, p. 168–180, apr 2018. ISSN 0736-5845. Disponível em: <<https://doi.org/10.1016/j.rcim.2017.09.005>>.

- MCKERROW, P. J. Robotics, an academic discipline? *Robotics*, v. 2, n. 3, p. 227–232, 1986. ISSN 0167-8493. Disponível em: <<https://www.sciencedirect.com/science/article/pii/0167849386900355>>.
- MICHEL, O. Webotstm: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, v. 1, 03 2004.
- M.M.M. SARCAR K. MALLIKARJUNA RAO, K. L. N. *Computer Aided Design and Manufacturing*. [S.l.: s.n.], 2008. 3 p. ISBN ISBN 9788120333420.
- MONTUFAR, G.; GHAZI-ZAHEDI, K.; AY, N. A theory of cheap control in embodied systems. *PLoS Computational Biology*, 09 2015.
- MORAES, E. N. *Método para gerenciamento do consumo de energia elétrica em sistemas ciberfísicos*. 57-59 p. Monografia (Doutorado) — Universidade Federal de Santa Catarina, Centro Tecnológico, Santa Catarina, 2013.
- NOF, S. Y. *Handbook of Industrial Robotics, Second*. 2 ed. [S.l.]: John Wiley e Sons, 1999.
- PAELKE, V. Augmented reality in the smart factory: Supporting workers in an industry 4.0. environment. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. [S.l.: s.n.], 2014. p. 1–4.
- PAREDE, I. M.; GOMES, L. E. L. *Eletrônica: automação industrial*. 6 ed. [S.l.]: São Paulo: Fundação Padre Anchieta, 2011. ISBN 978-85-61143-49-7.
- PEARSON, T. M. P. S. by. *An Introduction to the Kinect Sensor*. 7 2012. Url<https://www.microsoftpressstore.com/articles/article.aspx?p=2201646>.
- PISCHETOLA, M.; DE MIRANDA, L. Metodologias ativas: uma solução simples para um problema complexo? In: . [S.l.: s.n.], 2019.
- PYTHON SOFTWARE FOUNDATION, P. *General Python*. 2022. Url<https://www.python.org/about/apps/>.
- R. CRAWFORD, P. M. *Plastics Engineering*. 4 ed. [S.l.]: Elsevier, 2020. ISBN 9780081007099.
- REDDY, J. N. *SOLUTIONS MANUAL for An Introduction to The Finite Element Method (Third Edition)*. [s.n.], 2005. Disponível em: <<http://gen.lib.rus.ec/book/index.php?md5=8ce2417fd44703bd136cc1364da37b71>>.
- RITCHIE, D. M. The development of the c language. *ACM Sigplan Notices*, v. 28, n. 3, p. 201–208, 1993.
- ROHMER, E.; SINGH, S. P. N.; FREESE, M. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In: *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. [S.l.: s.n.], 2013. "www.coppeliarobotics.com".
- ROS, D. *ROS/Concepts*. 9 2022. Url<http://wiki.ros.org/ROS/Concepts>.

- ROSEN, R. et al. About the importance of autonomy and digital twins for the future of manufacturing. *IFAC-PapersOnLine*, v. 48, p. 567–572, 2015.
- RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. [S.l.]: Pearson Education, 2003.
- RÜSSMANN, M. et al. Industry 4.0 : The future of productivity and growth in manufacturing industries april 09. In: . [S.l.: s.n.], 2016.
- RUZZENENTE, M. et al. A review of robotics kits for tertiary education. In: . [S.l.: s.n.], 2012.
- RÉU JUNIOR, E. F. *Informática, redes e manutenção de computadores*. 2 ed. [S.l.]: São Paulo: Fundação Padre Anchieta, 2010. ISBN 978-85-61143-49-7.
- SACOMANO, J. et al. *Indústria 4.0: conceitos e fundamentos*. [S.l.]: Blucher, 2018. ISBN 9788521213710.
- SCHILDT, H. *C completo e total*. [S.l.]: Makron, 1997.
- SHIH, R. *Introduction to Finite Element Analysis Using SolidWorks Simulation 2014*. USA: SDC Publications, 2014. ISBN ISBN 1630573876.
- SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. *Introduction to Autonomous Mobile Robots*. 2nd ed. [S.l.]: The MIT Press, 2011. ISBN 0262015358.
- SIEGWART, R. Y.; NOURBAKHSH, I. R.; SCARAMUZZA, D. *Introduction to Autonomous Mobile Robots*. [S.l.: s.n.], 2004.
- SILVA, L. R. da. *ANÁLISE E PROGRAMAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS DA PLATAFORMA EYEBOT*. 37-38 p. Monografia (PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA) — Universidade Federal de Santa Catarina, Santa Catarina, 2003.
- SILVA ORTIGOZA, R. et al. Wheeled mobile robots: A review. *IEEE Latin America Transactions*, n. 6, p. 2209–2217, 2012.
- SINGH, M. et al. Digital twin: Origin to future. *Applied System Innovation*, v. 4, n. 2, 2021. ISSN 2571-5577. Disponível em: <<https://www.mdpi.com/2571-5577/4/2/36>>.
- SRINATH, K. R. Python – the fastest growing programming language. In: . [S.l.: s.n.], 2017.
- STALLINGS, W. *Arquitetura e organização de computadores*. 10. ed. ed. São Paulo: Pearson, 2017. ISBN 978-85-430-2053-2.
- STROUSTRUP, B. *An Overview of the C++ Programming Language*. [S.l.: s.n.], 1998.
- STUBBS, S. *Inclusive Education Where there are few resources*. 2 ed. [S.l.]: The Atlas Alliance, 2008.

SUBASI, A. Chapter 1 - introduction. In: SUBASI, A. (Ed.). *Practical Machine Learning for Data Analysis Using Python*. Academic Press, 2020. p. 1–26. ISBN 978-0-12-821379-7. Disponível em: <<https://www.sciencedirect.com/science/article/pii/B9780128213797000011>>.

SUDANDHIRA VEERAN, R. et al. Coppeliasim: Adaptable modular robot and its different locomotions simulation framework. *Materials Today: Proceedings*, 02 2021.

SUGIHARA, K. *Autonomous Robots and Vehicles*. 2022. URL http://www2.hawaii.edu/~sugihara/course/ics101/slides_a/01.html.

SYSTEMS, D. *DISCOVER CATIA*. 2022. URL <https://www.3ds.com/products-services/catia/>.

TANENBAUM, A.; GONÇALVES, R.; CONSULARO, L. *Sistemas operacionais modernos*. Prentice-Hall do Brasil, 2003. ISBN 9788587918574. Disponível em: <<https://books.google.com.br/books?id=meCAGQAACAAJ>>.

THYER, G. E. *Computer Numerical Control of Machine Tools*. 2 ed. [S.l.]: Butterworth Heinemann, 1991. ISBN 978-0-7506-0119-1.

TURLEBOT. *TurtleBot 3*. 2022. URL <https://www.turtlebot.com/turtlebot3/>.

TZAFESTAS, S. *Introduction to Mobile Robot Control*. USA: Elsevier, 2013. ISBN ISBN 9780124171039.

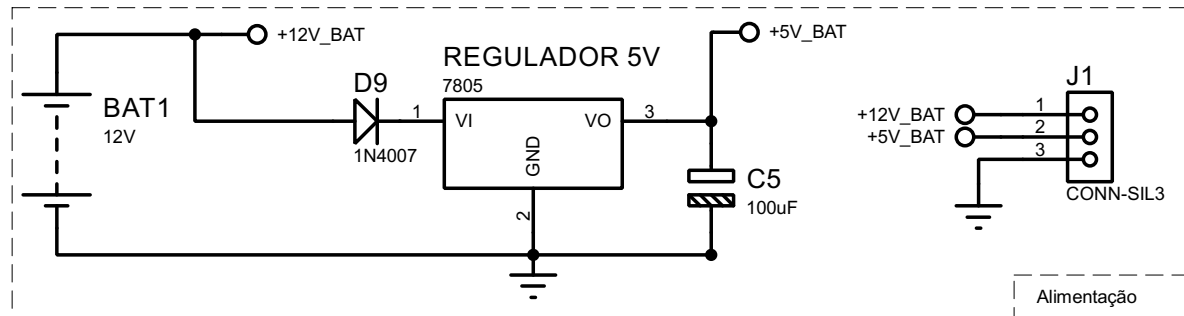
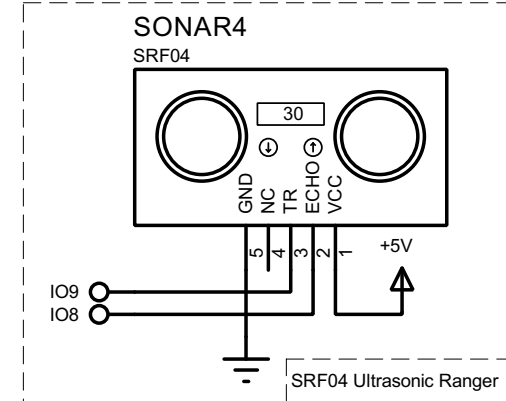
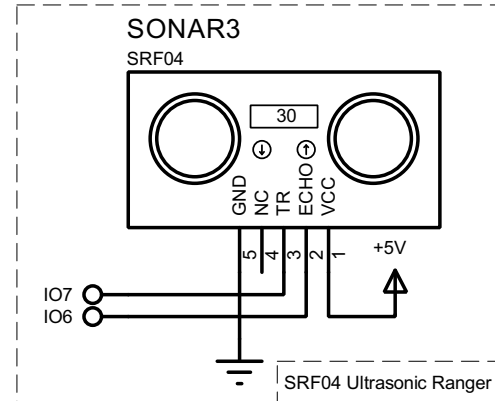
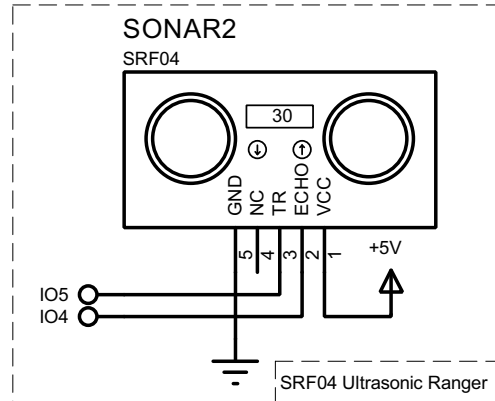
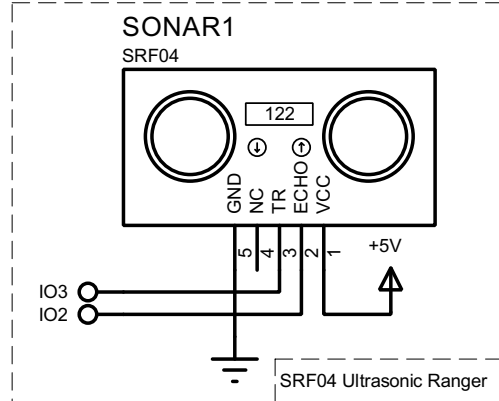
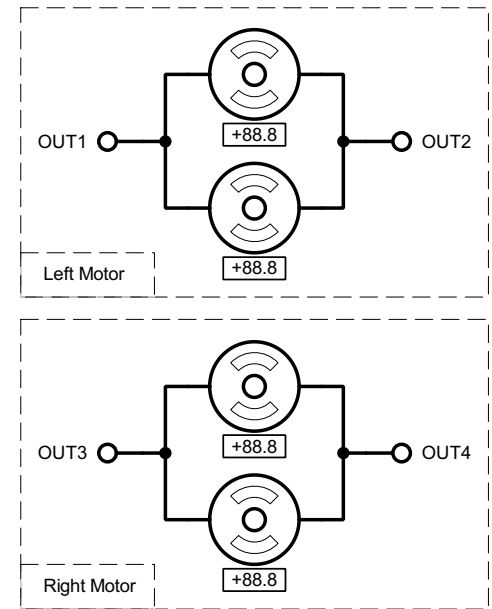
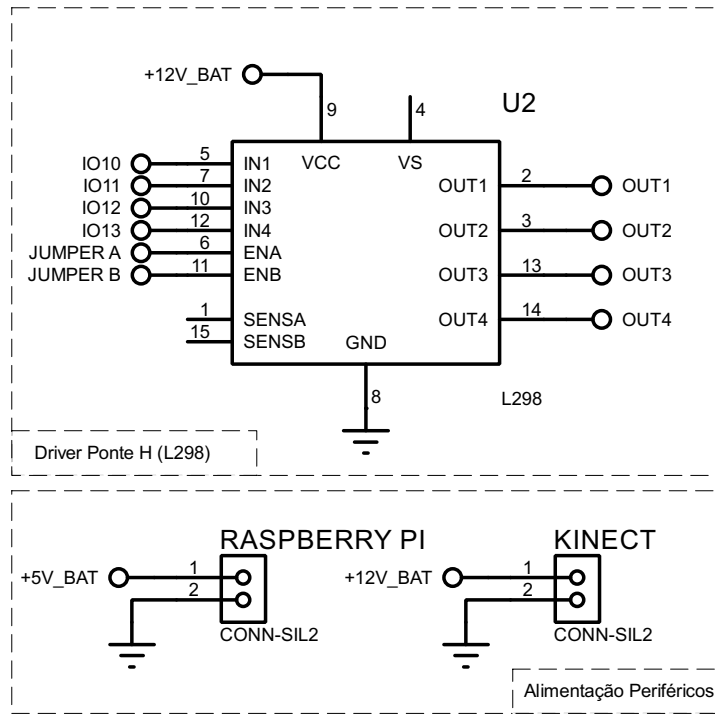
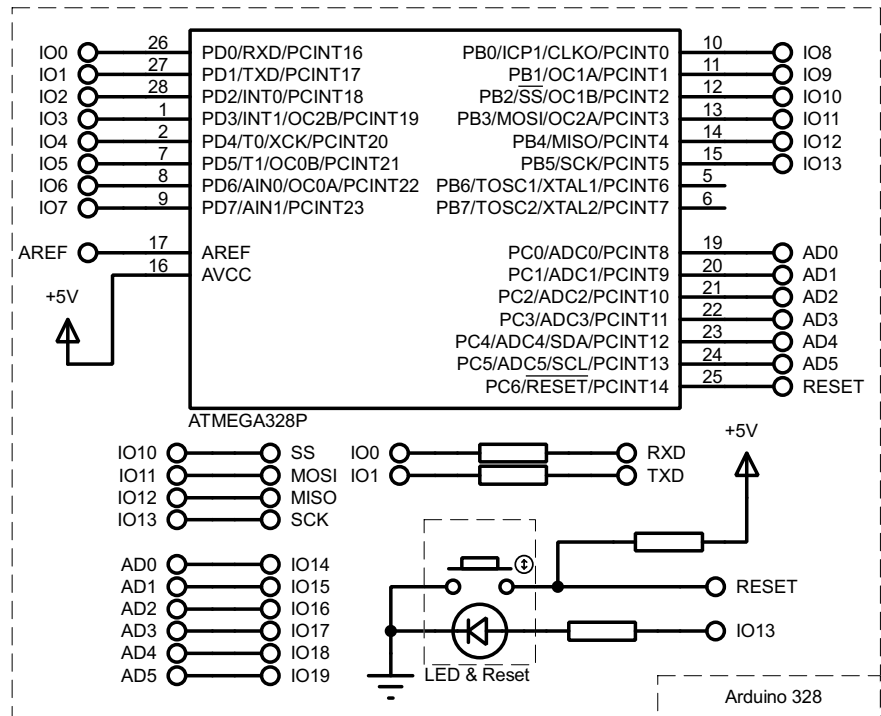
VIRNES, M. *Four Seasons of Educational Robotics: Substantive Theory on the Encounters between Educational Robotics and Children in the Dimensions of Access and Ownership*. Tese (Doutorado) — University of Eastern Finland, 12 2014.

WANG, G. *Introduction to CAD/CAE*. 2017. University Lecture.

XU, Z.; BAOJIE, X.; GUOXIN, W. Canny edge detection based on open cv. In: *2017 13th IEEE International Conference on Electronic Measurement and Instruments (ICEMI)*. [S.l.: s.n.], 2017. p. 53–56.

YANG, G.-Z. et al. The grand challenges of <i>science robotics</i>. *Science Robotics*, v. 3, n. 14, p. eaar7650, 2018. Disponível em: <<https://www.science.org/doi/abs/10.1126/scirobotics.aar7650>>.

APÊNDICE A - Esquemático Proteus Integração Física



APÊNDICE B – Programa Arduino Ponte H com ROS

```
#include <ros.h> //dedicada à comunicação dos pacotes ROS
#include <ros/time.h> //responsável pela sincronização dos
tempos ROS
#include <geometry_msgs/Twist.h> //trata do tipo de mensagem
ROS
```

```
#define IN1 10 //define que o pino 10 receberá o valor da
variável IN1
#define IN2 11 //define que o pino 11 receberá o valor da
variável IN2
#define IN3 12 //define que o pino 12 receberá o valor da
variável IN3
#define IN4 13 //define que o pino 13 receberá o valor da
variável IN4
```

```
void onTwist(const geometry_msgs::Twist& msg)
{
    if(msg.linear.x > 0) //TUPY PARA FRENTE
    {
        digitalWrite(IN1,HIGH); //roda direita para frente ON
        digitalWrite(IN2,LOW); //roda direita para trás OFF
        digitalWrite(IN3,HIGH); //roda esquerda para frente ON
        digitalWrite(IN4,LOW); //roda esquerda para trás OFF
    }
    else if(msg.linear.x < 0) //TUPY PARA TRÁS
    {
        digitalWrite(IN1,LOW); //roda direita para frente OFF
        digitalWrite(IN2,HIGH); //roda direita para trás ON
        digitalWrite(IN3,LOW); //roda esquerda para frente OFF
        digitalWrite(IN4,HIGH); //roda esquerda para trás ON
    }
    else if(msg.angular.z < 0) //TUPY GIRA PARA ESQUERDA
    {
        digitalWrite(IN1,HIGH); //roda direita para frente ON
        digitalWrite(IN2,LOW); //roda direita para trás OFF
        digitalWrite(IN3,LOW); //roda esquerda para frente OFF
        digitalWrite(IN4,HIGH); //roda esquerda para trás ON
    }
    else if(msg.angular.z > 0)//TUPY GIRA PARA DIREITA
    {
        digitalWrite(IN1,LOW); //roda direita para frente OFF
        digitalWrite(IN2,HIGH); //roda direita para trás ON
        digitalWrite(IN3,HIGH); //roda esquerda para frente ON
        digitalWrite(IN4,LOW); //roda esquerda para trás OFF
    }
    else //PARAR TUPY
```

APÊNDICE C – Programa Arduino 4
Sensores Ultrassônicos com
ROS

```

#include <ros.h> //dedicada à comunicação dos pacotes ROS
#include <ros/time.h> //responsável pela sincronização dos
tempos ROS
#include <sensor_msgs/Range.h> //trata do tipo de mensagem
ROS
#include <NewPing.h> //biblioteca dedicada ao uso de sonares
#include <SimpleKalmanFilter.h> //biblioteca de filtros
digitais

#define SONAR_NUM 4          //Número de sonares
#define MAX_DISTANCE 200    //Maxima distância de detecção
dos obstáculos
#define PING_INTERVAL 33    //Loop do ping a cada 33
microsegundos.

unsigned long pingTimer[SONAR_NUM]; //tempo de ping para cada
sensor.
unsigned int cm[SONAR_NUM];          // Onde as distâncias de
ping são armazenadas.
uint8_t currentSensor = 0;          // Mantém o controle de
qual sensor está ativo.
unsigned long _timerStart = 0;      //tempo inicia em 0
int LOOPING = 40;                  //Loop a cada 40
milissegundos.
uint8_t oldSensorReading[3];        //Armazena o último valor
válido dos sensores.

uint8_t oneSensor;                  //Armazena o valor real do sensor
1.
uint8_t twoSensor;                  //Armazena o valor real do sensor
2.
uint8_t threeSensor;                //Armazena o valor real do sensor
3.
uint8_t fourSensor;                 //Armazena o valor real do sensor
4.

uint8_t oneSensorKalman;            //Armazena o valor filtrado do
sensor 1.
uint8_t twoSensorKalman;            //Armazena o valor filtrado do
sensor 2.
uint8_t threeSensorKalman;          //Armazena o valor filtrado do
sensor 3.
uint8_t fourSensorKalman;           //Armazena o valor filtrado do
sensor 4.

NewPing sonar[SONAR_NUM] = //cria objetos newPing para todos

```

APÊNDICE D – Programa Arduino Completo Robô Tupy


```

#include <ros.h> //dedicada à comunicação dos pacotes ROS da
bib. roserial-arduino
#include <ros/time.h> //responsável pela sincronização dos
tempos ROS da bib. roserial-arduino
#include <geometry_msgs/Twist.h> //trata vetores lineares a
angulares ROS da bib. roserial-arduino
#include <sensor_msgs/Range.h> //função ROS que trata da
distância sonares da bib. roserial-arduino

#include <NewPing.h> //biblioteca dedicada ao uso de sonares
#include <SimpleKalmanFilter.h> //biblioteca de filtros
digitais

#define IN1 10 //define que o pino 10 receberá o valor da
variável IN1
#define IN2 11 //define que o pino 11 receberá o valor da
variável IN2
#define IN3 12 //define que o pino 12 receberá o valor da
variável IN3
#define IN4 13 //define que o pino 13 receberá o valor da
variável IN4

#define SONAR_NUM 4 //Número de sonares
#define MAX_DISTANCE 200 //Maxima distância de detecção
dos obstáculos
#define PING_INTERVAL 33 //Loop do ping a cada 33
microsegundos.

unsigned long pingTimer[SONAR_NUM]; //tempo de ping para
cada sensor.
unsigned int cm[SONAR_NUM]; // Onde as
distâncias de ping são armazenadas.
uint8_t currentSensor = 0; // Mantém o controle
de qual sensor está ativo.
unsigned long _timerStart = 0; //tempo inicia em 0
int LOOPING = 40; //Loop a cada 40
milissegundos.
uint8_t oldSensorReading[3]; //Armazena o último
valor válido dos sensores.

uint8_t oneSensor; //Armazena o valor real do sensor
1.
uint8_t twoSensor; //Armazena o valor real do sensor
2.
uint8_t threeSensor; //Armazena o valor real do sensor
3.

```

APÊNDICE E – Programa em Python 2.7 Canny Kinect

```
#!/usr/bin/python2.7
#DIGITAL TWIN MOBILE ROBOT
#Modelo processamento de imagens para OpenCV - Canny Edge Detection
import rospy
from sensor_msgs.msg import Image
import cv2
import numpy as np #biblioteca para computacao cientifica
import time

#ROS image message -> OpenCV2 image converter
from cv_bridge import CvBridge, CvBridgeError #pacote do ROS com diferentes tipos
de formatacao de imagens

#for delay measure
delay=0
delta=0
bridge = CvBridge()

import sys
import signal
def signal_term_handler(signal, frame):
    rospy.logger('User KeyboardInterrupt')
    sys.exit(0)
signal.signal(signal.SIGINT, signal_term_handler)

#callback function funcao principal
def img_callback(ros_data):
    global delta
    print(int(round(time.time()*1000)) - delta)
    delta = int(round(time.time()*1000))

    try:
        image_np = bridge.imgmsg_to_cv2(ros_data, "mono8") #tipagem do
open cv rgb
        #image_np = bridge.imgmsg_to_cv2(ros_data, "bgr8") #tipagem do
open cv em escala de cinza
        flipVertical_image_np = cv2.flip(image_np, 1)
    except CvBridgeError as e:
        print(e)

    edges = cv2.Canny(image_np, 50, 150, apertureSize=3) #basicamente um novo
frame. opera em escala de cinza
    flipVertical_edges = cv2.flip(edges, 1)

    #cv2.imshow("output", image_np)
    #cv2.imshow("output", np.hstack([image_np, edges])) #concatenacao de
matrizes
    cv2.imshow("output", np.hstack([flipVertical_image_np,
flipVertical_edges]))
    cv2.waitKey(1) #sleep de 1ms

def main():
    rospy.init_node('csf_robotCS_cannyKinect')
    rospy.Subscriber("camera/rgb/image_raw", Image, img_callback, queue_size=1)
    #rospy.Subscriber("/kinect/rgb_image", Image, img_callback, queue_size=1)

    try:
        rospy.spin()
    except KeyboardInterrupt:
        print('Shutting down ...')
    cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```