



UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CENTRO TECNOLÓGICO  
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Raul Victor Trombin

**Desenvolvimento de uma Biblioteca Multiplataforma em Rust para Veículos  
Remotamente Controlados: Integração de Sensores e Atuadores**

Florianópolis

2023

Raul Victor Trombin

**Desenvolvimento de uma Biblioteca Multiplataforma em Rust para Veículos  
Remotamente Controlados: Integração de Sensores e Atuadores**

Trabalho de Conclusão de Curso do curso de Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharel em Engenharia Elétrica.

Orientador(a): Prof. Jean Viane Leite, Dr.

Florianópolis

2023

Trombin, Raul Victor

Desenvolvimento de uma Biblioteca Multiplataforma em Rust para Veículos Remotamente Controlados: Integração de Sensores e Atuadores / Raul Victor Trombin ; orientador, Jean Viane Leite, 2023.

75 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2023.

Inclui referências.

1. Engenharia Elétrica. 2. ROV. 3. Rust. 4. Sistemas Embarcados. I. Leite, Jean Viane . II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Raul Victor Trombin

**Desenvolvimento de uma Biblioteca Multiplataforma em Rust para Veículos Remotamente Controlados: Integração de Sensores e Atuadores**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel em Engenharia Elétrica e aprovado em sua forma final pelo curso de Engenharia Elétrica.

Florianópolis, 15 de dezembro de 2023.

Insira neste espaço  
a assinatura

Coordenação do Curso

**Banca examinadora**

Insira neste espaço  
a assinatura

Prof. Jean Viane Leite, Dr.

Orientador

Insira neste espaço  
a assinatura

Prof. Mauricio Valencia Ferreira da Luz, Dr.

UFSC

Insira neste espaço  
a assinatura

Eng. Eletric. Carlos Alexandre Corrêa Wengerkievicz, Dr.

UFSC

Dedico este trabalho aos meus pais e ao meu irmão.

## **AGRADECIMENTOS**

Aos meus pais, sou imensamente grato pelo amor incondicional, orientação e sacrifícios. Vocês sempre acreditaram em mim e me encorajaram a perseguir meus sonhos. A confiança e o apoio constantes foram fundamentais para eu superar os desafios e alcançasse esta conquista.

Ao meu irmão, sou grato pela parceria, apoio e incentivo ao longo de todos esses anos. Suas palavras de encorajamento e momentos de descontração trouxeram leveza e alegria à minha jornada acadêmica.

À Isabela, agradeço por estar ao meu lado durante todo esse tempo. Sua compreensão, paciência e apoio incondicional foram essenciais para eu me manter motivado e focado nos momentos mais desafiadores. Sua presença trouxe equilíbrio e felicidade à minha vida.

Ao meu orientador, professor Jean Viane Leite, pela abordagem, organização e orientação no decorrer deste trabalho de conclusão de curso.

À equipe da BlueRobotics, composta também pelos meus antigos colegas do projeto Robota da UFSC, gostaria de expressar minha gratidão pela colaboração e orientação durante o desenvolvimento desse trabalho. As experiências e a dedicação de vocês na inovação no campo dos ROVs foram inestimáveis e contribuíram significativamente para o sucesso deste projeto.

Aos meus colegas de faculdade, sou grato pela amizade, colaboração e pelo compartilhamento de experiências acadêmicas. Nossas discussões e momentos de aprendizado conjunto foram fundamentais para meu crescimento pessoal e profissional.

À UFSC e a todo seu corpo docente e demais servidores, agradeço pelo comprometimento em buscar e manter a excelência na educação.

A todos aqueles que não pude mencionar, obrigado pelo amor, apoio e confiança. Sou verdadeiramente abençoado por ter minha família e um círculo de pessoas tão incríveis ao meu redor.

"Compreendo agora o sentido das coisas últimas e extremas que podem ser expressas em pensamento, poesia - e em fé humana: a redenção pelo amor e no amor." (Viktor Frankl, 2020, p.55)

## RESUMO

Este trabalho explora o desenvolvimento de uma biblioteca robusta e multiplataforma para Veículos Operados Remotamente (ROVs) usando a linguagem de programação Rust, além de disponibilizar bibliotecas para C++ e Python. Com foco no hardware Navigator – um acessório de Raspberry Pi e componente do submarino BlueROV da empresa BlueRobotics – realizou-se um estudo para a integração dos diferentes sensores que o compõe. Destaca-se que a sua integração foi realizada em uma sintaxe acessível, de forma a facilitar o uso da biblioteca pelas abstrações possíveis. Além disso, desenvolveu-se a solução para diferentes arquiteturas de processador ARM: armv7 e aarch64. Explorou-se também a compatibilidade com musllinux e manylinux, padrões que garantem a portabilidade dos binários para diferentes sistemas operacionais, como Raspbian e Alpine. Ao longo do trabalho, abordam-se os desafios da compilação multiplataforma, o processo da criação da biblioteca em Rust até a sua publicação e a criação de sua portabilidade para Python e C++. A implantação e o processo de integração contínua/entrega contínua (CI/CD) são criados por meio do GitHub Actions, o que permitiu o avanço do trabalho com códigos consistentes, testados e organizados.

**Palavras-chave:** ROV; Rust; Sistemas Embarcados.



## ABSTRACT

This work deals with the development of a robust and cross-platform library for Remotely Operated Vehicles (ROVs) using the Rust programming language, with additional libraries provided for C++ and Python. Focusing on the Navigator hardware – a Raspberry Pi accessory and a component of the BlueROV submarine by BlueRobotics Company– a study was conducted to integrate its various sensors. The integration was performed with an accessible syntax, aiming to facilitate the library's usage through possible abstractions. Additionally, solutions were developed for different ARM processor architectures: armv7 and aarch64. Compatibility with musllinux and manylinux was also explored, adhering to standards that ensure binary portability across various operating systems such as Raspbian and Alpine. Throughout the work, challenges of cross-platform compilation are addressed, covering the process of library creation in Rust to its publication and the establishment of portability for Python and C++. Deployment and continuous integration/continuous delivery (CI/CD) processes are implemented through GitHub Actions, enabling the advancement of the project with consistent, tested, and organized code.

**Keywords:** ROV; Rust; Embedded Systems.

## LISTA DE FIGURAS

Figura 1 – Foto do veículo CURV II.....	21
Figura 2 – ROV Kaiko colocando uma bandeira no local mais profundo da Terra ....	22
Figura 3 – Classificação dos veículos subaquáticos .....	23
Figura 4 – Imagem da constelação de veículos do projeto ARGO.....	24
Figura 5 – Imagem do ROV Jason na erupção do West Mata .....	25
Figura 6 – Imagens do projeto CUREE identificando espécies.....	26
Figura 7 – BlueROV2 da BlueRobotics em configuração padrão.....	27
Figura 8 – Gráfico destacando a progressão da linguagem Rust no Android .....	29
Figura 9 – Preferência de Linguagens de Programação na Pesquisa de Desenvolvedores do Stack Overflow.....	30
Figura 10 – Fluxo de trabalho no desenvolvimento.....	35
Figura 11 – Entendendo o GitHub Action.....	35
Figura 12 – A placa Navigator conectada a um Raspberry Pi 4 .....	37
Figura 13 – Thruster T200 da BlueRobotics.....	41
Figura 14 – Lumen Subsea .....	41
Figura 15 – Lumen Light em uso.....	41
Figura 16 – Cápsulas submarinas.....	46
Figura 17 – Resultados da execução do código no terminal.....	50
Figura 18 – Fluxo de ações no GitHub Actions .....	54
Figura 19 – Publicação do navigator-rs no crates.io .....	54
Figura 20 – Documentação do navigator-rs .....	55
Figura 21 – Fluxo de ações no GitHub Actions .....	59
Figura 22 – Biblioteca Navigator para Python .....	59
Figura 23 – Página com documentação para biblioteca Python.....	60
Figura 24 – Exemplo de Extensão do BlueOS utilizando a biblioteca Python (a).....	61
Figura 25 – Exemplo de Extensão do BlueOS utilizando a biblioteca Python (b).....	61
Figura 26 – Exemplo de Extensão do BlueOS utilizando a biblioteca Python (c).....	62
Figura 27 – Diagrama listando os projetos desenvolvidos e suas dependências.....	63
Figura 28 – Anúncio das bibliotecas na rede social LinkedIn .....	64
Figura 29 – Anúncio da biblioteca no fórum principal (a) .....	65
Figura 30 – Anúncio da biblioteca no fórum principal (b) .....	65

Figura 31 – Anúncio da biblioteca no fórum principal (c).....	65
Figura 32 – Anúncio da biblioteca no fórum principal (d) .....	66
Figura 33 – Anúncio da biblioteca no fórum principal (e) .....	66

## LISTA DE EQUAÇÕES

Equação 1 – Cálculo do PRE\_SCALE .....**Erro! Indicador não definido.**

## LISTA DE TABELAS

Tabela 1 – Tabela das versões disponíveis Python .....	33
Tabela 2 – Tabela listando os componentes da Navigator.....	38
Tabela 3 – Funções alternativas por GPIO no Raspberry Pi 4.....	39

## LISTA DE GRÁFICOS

Gráfico 1 – Gráficos gerados pelo Criterion – Leitura do magnetômetro .....	52
Gráfico 2 – Gráficos gerados pelo Criterion – Leitura de todos sensores .....	53

## LISTA DE CÓDIGOS

Código 1 – Exemplo da biblioteca utilizando modulo PWM .....	42
Código 2 – Exemplo da biblioteca utilizando modulo ICM20689.....	43
Código 3 – Exemplo da biblioteca utilizando modulo ICM20689 b.....	43
Código 4 – Exemplo da biblioteca utilizando modulo ADS1115.....	44
Código 5 – Exemplo da biblioteca utilizando modulo AK09915.....	45
Código 6 – Exemplo da biblioteca utilizando módulo BMP280.....	46
Código 7 – Exemplo da biblioteca utilizando modulo BMP280.....	46
Código 8 – Exemplo da biblioteca utilizando os LEDs .....	47
Código 9 – Exemplo da biblioteca utilizando o neopixel.....	48
Código 10 – Código com estrutura do objeto Navigator.....	48
Código 11 – Código exemplo .....	49

## LISTA DE ABREVIATURAS E SIGLAS

- ADC – Conversor Analógico-Digital, do inglês *Analog-to-Digital Converter*.
- API – Interface de Programação de Aplicações, do inglês *Application Programming Interface*.
- AUVS – Veículos Autônomos Subaquáticos, do inglês *Autonomous Underwater Vehicle*.
- CI – Circuito Integrado, do inglês *Integrated Circuit*.
- CI/CD – Integração Contínua/Entrega Contínua, do inglês *Continuous Integration/Continuous Delivery*.
- CUREE – em tradução livre para o português Robô Curioso para Exploração de Ecossistema, do inglês *Curious Robot For Ecosystem Exploration*.
- CURV – Veículo de Recuperação Submarina Controlada a Cabo, do inglês *Cable-Controlled Underwater Recovery Vehicle*.
- DSL – Laboratório de Submersão Profunda, do inglês *Deep Submergence Laboratory*.
- FFI – Interface de Função Externa, do inglês *Foreign Function Interface*.
- GPIO – Entrada/Saída para Propósitos Gerais, do inglês *General Purpose Input/Output*.
- HAL – Camada de Abstração de Hardware, do inglês, *Hardware Abstraction Layer*.
- HAT – em tradução livre para o português Hardware Anexado na Parte Superior, do inglês *Hardware Attached on Top*.
- OE – em tradução livre para o português, Habilitação de Saída, do inglês *Output-Enable*.
- PWM – Modulação por Largura de Pulso, do inglês *Pulse Width Modulation*.
- PYPI – em tradução livre para o português Integração de Pacotes Python, do inglês *Python Packages Integration*.
- ROVS – Veículos Operados Remotamente, do inglês *Remotely Operated Underwater Vehicle*.
- SDK – Kit de desenvolvimento de software, do inglês *Software Development Kit*.
- UUVs – Veículos Subaquáticos Não Tripulados, do inglês *Unmanned Underwater Vehicles*.
- WHOI – em tradução livre para português, Instituto Oceanográfico de Woods Hole, do inglês *Woods Hole Oceanographic Institution*,





## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b> .....	<b>16</b>
<b>2</b>	<b>OBJETIVOS</b> .....	<b>18</b>
2.1	OBJETIVO GERAL .....	18
2.2	OBJETIVOS ESPECÍFICOS .....	19
<b>3</b>	<b>ROVS</b> .....ERRO! INDICADOR NÃO DEFINIDO.	
3.1	UMA BREVE HISTÓRIA DOS ROVS .....	20
3.2	CLASSIFICAÇÃO DOS VEÍCULOS AQUÁTICOS.....	22
3.3	PROJETOS QUE UTILIZAM ROVS.....	24
<b>3.3.1</b>	<b>Projeto ARGO</b> .....	<b>24</b>
<b>3.3.2</b>	<b>ROV Jason, da WHOI</b> .....	<b>25</b>
<b>3.3.3</b>	<b>Projeto CUREE</b> .....	<b>25</b>
<b>3.3.4</b>	<b>BlueROV2</b> .....	<b>26</b>
<b>4</b>	<b>LINGUAGEM RUST</b> .....	<b>28</b>
4.1	RUST NO PROJETO ANDROID.....	28
4.2	RUST NA MICROSOFT.....	29
4.3	STACK OVERFLOW.....	29
<b>5</b>	<b>DESENVOLVIMENTO - RUST</b> .....	<b>31</b>
5.1	RUST – CROSS-COMPILING .....	31
5.2	RUST – FFI.....	32
<b>5.2.1</b>	<b>FFI e C++</b> .....	<b>32</b>
<b>5.2.2</b>	<b>FFI e Python</b> .....	<b>33</b>
<b>6</b>	<b>DESENVOLVIMENTO - SISTEMA DE VERSIONAMENTO E CI/CD.</b> .....	<b>35</b>
<b>7</b>	<b>DESENVOLVIMENTO - PROJETO NAVIGATOR-RS</b> .....	<b>37</b>
7.1	ESTUDO E INTEGRAÇÃO DO HARDWARE .....	38
<b>7.1.1</b>	<b>O Modulo PWM</b> .....	<b>40</b>
<b>7.1.2</b>	<b>Acelerômetro E Giroscópio</b> .....	<b>42</b>
<b>7.1.3</b>	<b>Conversor analógico digital (ADC)</b> .....	<b>43</b>
<b>7.1.4</b>	<b>Magnetômetro</b> .....	<b>44</b>
<b>7.1.5</b>	<b>Sensor de temperatura e pressão</b> .....	<b>45</b>
<b>7.1.6</b>	<b>LEDs de Status e Neopixel</b> .....	<b>47</b>
7.2	INTEGRAÇÃO .....	48

7.3	BENCHMARK E TESTES.....	52
7.4	CI/CD INTEGRAÇÃO.....	53
7.5	PUBLICAÇÃO.....	54
<b>8</b>	<b>DESENVOLVIMENTO – O PROJETO NAVIGATOR-LIB.....</b>	<b>56</b>
8.1	PYTHON.....	56
<b>8.1.1</b>	<b>Projeto.....</b>	<b>57</b>
<b>8.1.2</b>	<b>Documentação.....</b>	<b>57</b>
8.2	C++.....	57
<b>8.2.1</b>	<b>Compilação.....</b>	<b>57</b>
<b>8.2.2</b>	<b>Documentação.....</b>	<b>58</b>
8.3	CI/CD INTEGRAÇÃO.....	58
8.4	PUBLICAÇÃO.....	59
8.5	EXTENSÃO DO BLUEOS.....	60
<b>9</b>	<b>RESULTADOS OBTIDOS.....</b>	<b>63</b>
<b>10</b>	<b>CONCLUSÕES.....</b>	<b>67</b>

## 1 INTRODUÇÃO

A Terra é o único planeta conhecido por ter a água em forma líquida em sua superfície. Em nosso planeta, 70,8% da superfície terrestre é coberta por água e 97% da água está nos oceanos. Deste percentual da superfície oceânica, o Pacífico é responsável por 32,4%, seguido do Atlântico e Índico, com 16,2% e 14,4% respectivamente (Christ; Wernli, 2018). A água dissolve elementos em grandes quantidades e desempenha um papel crucial na manutenção do clima e da vida no planeta.

A profundidade média dos oceanos da Terra é de 3,9 km (Smith, 1997). O local com a maior profundidade mensurada até hoje é na Fossa das Marianas, localizada no Oceano Pacífico, com um registro de 10.920 metros (Nakanishi; Hashimoto, 2011). Estima-se que menos de 24,9% do fundo do oceano tenha sido mapeado até então (Mayer *et al.*, 2018; Seabed 2030), sendo a pesquisa contínua dessas profundezas rica em informações sobre a geologia, a biodiversidade e os processos oceânicos, que contribuem para uma compreensão global dos oceanos e de seu papel no sistema terrestre.

A utilização de Veículos Subaquáticos Operados Remotamente (ROVs, do inglês *Remotely Underwater Operated Vehicles*) permite operações em locais inacessíveis à presença humana, como operações em áreas de risco ou exploração espacial e marítima. Esses equipamentos permitem, pois, a realização de trabalhos, como a manipulação de objetos e inspeção visual, de forma remota e segura.

Os custos financeiros e os riscos associados à operação com seres humanos em mar profundo podem ser expressivos. Em comparação aos ROVs, a operação humana requer pausas eventuais devido à pressão no ato de mergulho. Esses fatores tornam ainda mais atraente a busca e o emprego de ROVs na indústria e em pesquisas científicas. Além disso, destaca-se o uso de ROVs em operações de busca e resgate em condições desafiadoras à presença humana, como águas turvas e de baixa visibilidade. Em se tratando de busca por pessoas, por exemplo, em que o tempo é um fator crítico, os ROVs tornam-se uma ferramenta crucial para o êxito da operação (FISHERS, 2021).

ROVs são sistemas que ficam submersos e podem sofrer altas pressões dependendo de sua categoria e profundidade de trabalho. Precisam de invólucros para sua proteção e de sistemas robustos e confiáveis para suas mensurações e controle.

É possível dividir os ROVs em duas categorias: os remotamente operados e os autônomos. Ambos precisam de uma plataforma com hardware embarcado e interfaces que realizem a abstração do hardware embarcado para o seu devido funcionamento. Tais plataformas possuem sistemas embarcados com sensores de diferentes tipos, que auxiliam na navegação, além de outros periféricos atuadores, que realizam o controle e a manipulação. Para a utilização desses periféricos, é essencial o uso de bibliotecas — códigos para facilitar a comunicação com o hardware — de maneira a tornar a realização de aplicações em software mais fáceis de serem desenvolvidas e gerenciadas. Este ecossistema é popularmente conhecido como Kit de Desenvolvimento de Software (SDK, do inglês *Software Development Kit*), ou seja, um pacote contendo todos os códigos necessários para desenvolvedores criarem suas aplicações.

Neste contexto, insere-se a empresa BlueRobotics, que fornece submarinos e barcos ROVs acessíveis, tanto para estudantes quanto para empresas do ramo, além de uma plataforma com sensores e atuadores integradas, a Navigator.

O presente trabalho expõe a solução criada para disponibilizar bibliotecas para a plataforma Navigator. Com a plataforma, foi possível desenvolver um sistema de entrega contínua e disponibilizar bibliotecas para três linguagens populares: C++, Python e Rust, sendo esta última a linguagem central deste projeto. Ressalta-se que a linguagem Rust não oferece apenas uma sintaxe moderna, mas também uma plataforma inteira que fornece ao programador um revisor de código, um compilador com assistência, uma verificação de boas práticas e um suporte para geração automática de documentação.

O projeto foi desenvolvido utilizando uma plataforma de versionamento de códigos, GitHub, além de usar um sistema de controle de qualidade e entrega — *continuous integration/continuous delivery* (CI/CD, em tradução livre para o português, Integração Contínua/Entrega Contínua) — utilizando o GitHub Actions.

## 2 OBJETIVOS

Um ROV precisa, idealmente, de uma plataforma que integre sensores e atuadores para sua operação. Essa plataforma é colocada dentro do equipamento, e é coberta por um invólucro que a protege das intempéries das águas.

No mercado, existem diversos fabricantes e plataformas que possibilitam aos desenvolvedores a integração com sensores. Essa integração permite a medição de diversas grandezas como aceleração, campo magnético, temperatura e pressão. Também são possíveis integrações para o controle de motores, iluminação, manipuladores e transmissão de vídeo e outros dados.

Algumas destas plataformas também possuem um microcomputador embarcado, o que oferece uma capacidade maior de processamento para testar as funcionalidades do ROV. Um exemplo desta plataforma é a Navigator, que é fornecida pela empresa BlueRobotics.

A plataforma de sistemas embarcados Navigator tem um design compatível com o Raspberry Pi 4 no formato conhecido como *hardware attached on top* (HAT, em tradução livre para português, Hardware Anexado na Parte Superior). A placa Raspberry Pi 4 possui um processador compatível com a arquitetura ARM, que é muito comum em diversos outros *chipsets* como fabricantes de celulares e, mais recentemente, computadores, como Macbook com processador M1 e M2. O *chipset* BC2721 é compatível com as arquiteturas de software arm7 e aarch64, o que permite o usuário a optar entre diferentes sistemas operacionais disponíveis.

### 2.1 OBJETIVO GERAL

O propósito deste trabalho é apresentar uma solução para manipular o *hardware* Navigator. Para isso, é necessário criar uma biblioteca, na qual objetiva-se aplicar boas práticas conhecidas no mercado, como por exemplo: organização, versionamento, documentação e automação de testes e distribuição.

A biblioteca, objeto de estudo do presente trabalho, é voltada não apenas aos ROVs da BlueRobotics, mas a qualquer desenvolvedor que busque a Navigator como plataforma para seu projeto. Além disso, ressalta-se a necessidade de criar uma

solução que abranja outras linguagens e arquiteturas com o objetivo de desenvolver uma biblioteca que possa ser utilizada pelo maior número de desenvolvedores.

## 2.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos se dividem em:

- Realizar uma análise sobre a ciência dos ROV;
- Explorar sobre a linguagem Rust e demais integrações;
- Analisar a plataforma Navigator da empresa BlueRobotics;
- Estudar sobre *binding* e bibliotecas para outras linguagens;
- Propor e disponibilizar uma biblioteca para desenvolvedores.

### 3 VEICULOS REMOTAMENTE OPERADOS

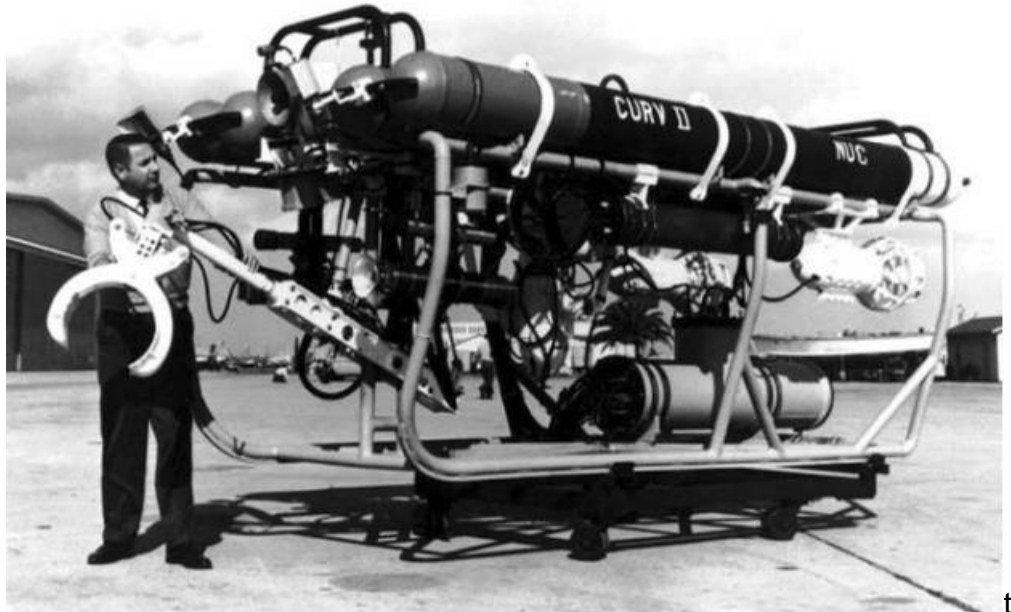
Os ROVs são ferramentas de extrema importância para o mundo moderno. O seu uso abrange o monitoramento de mudanças climáticas, detecção e estudo de catástrofes, identificação e acompanhamento de microrganismos, como cianobactérias e fitoplâncton, observação de espécies em extinção e estudo de organismos aquáticos, como corais. Na engenharia, o uso de ROVs é observado na construção e inspeção de instalações submarinas, portos, inspeção de cascos e hélices de navios, armamento bélico naval, além do acompanhamento de vazamentos de óleo em águas profundas.

#### 3.1 UMA BREVE HISTÓRIA DOS ROVS

É atribuído a Dimitri Rebikoff (1921-1997) o desenvolvimento do primeiro ROV que se tem conhecimento, no ano de 1953. Batizado de Poodle, o veículo foi usado em operações com torpedos. As pesquisas publicadas por Rebikoff e seus inventos na esfera de tecnologias submarinas propiciaram sua migração para os Estados Unidos da América (EUA) e a colaboração em projetos secretos da Marinha dos EUA (Corrêa, 2022). Foi, então, na década de 60 que os EUA se destacaram no desenvolvimento de tecnologia referente ao ROV, incluindo um equipamento denominado Veículo de Recuperação Submarina Controlada a Cabo (CURV, do inglês *Cable-Controlled Underwater Recovery Vehicle*), o qual ganhou notoriedade após recuperar uma bomba atômica perdida em um acidente de avião na costa de Palomares em 1966 (Ferreira; Maravilha, 2008). Uma versão desse veículo pioneiro é vista na Figura 1.



Figura 1 – Foto do veículo CURV II



Fonte: Christ e Wernli (2007, p.1112).

O investimento em ROVs teve um crescimento significativo em meados das décadas de 1970 e 1980, além de uma mudança no perfil de investidores. De 1953 a 1974, 85% dos veículos construídos tinham como origem investimentos governamentais. De 1974 a 1982, o investimento privado aumentou de maneira exponencial, sendo responsável pela construção de 96% dos veículos desse gênero (Christ; Wernli, 2018).

Nos anos seguintes, de 1982 a 1989, novamente, houve um grande investimento na área dos ROVs, e, por consequência, se realizou a primeira conferência sobre este tema em ascensão, a qual tinha como temática “Uma tecnologia cuja hora chegou”. Em termos de comparação com a indústria privada, em 1970, havia apenas uma empresa no ramo dos veículos operados remotamente e quatorze anos depois esse número saltou para vinte e sete.

Os ROVs surgiram com a explosão de uma era tecnológica e firmaram espaço no que tange a diferentes áreas de serviço, pesquisa e lazer. É esperado que os gastos com essa vertente tecnológica continuem em expansão. Isso ocorre, principalmente, por seu potencial de adaptação às mais diversas áreas corporativas e

pela possibilidade de poder atuar em ambientes considerados hostis pelos humanos, como submergir a profundidades cujas pressões são totalmente adversas às condições humanas.

Em 1995 têm-se os primeiros registros do ROV Kaiko, um destes registros pode ser visto na Figura 2, em que se realiza a medição de 10.911m de profundidade na Fossa das Marianas.

Figura 2 – ROV Kaiko colocando uma bandeira no local mais profundo da Terra



Fonte: ©Jamstec (2015).

### 3.2 CLASSIFICAÇÃO DOS VEÍCULOS AQUÁTICOS

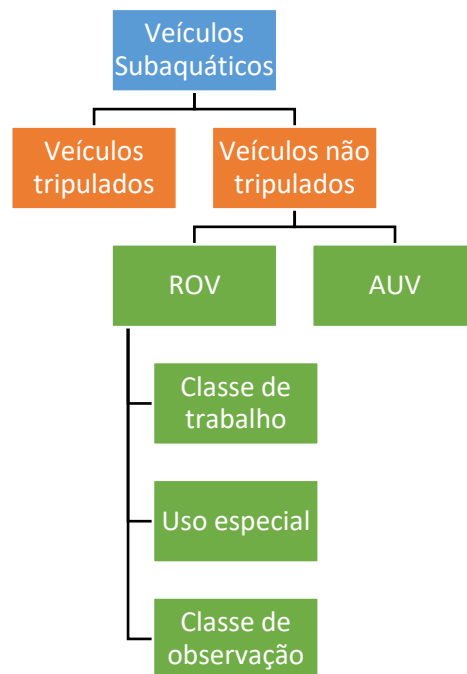
Atualmente, pode-se encaixar os veículos aquáticos em duas diferentes classes, sendo elas os Veículos Subaquáticos Tripulados e os Veículos Subaquáticos Não Tripulados (UUVs, do inglês *Unmanned Underwater Vehicles*)<sup>1</sup>. Na indústria, a

---

<sup>1</sup> Em específico, a Marinha dos Estados Unidos da América utiliza a classe UUV como sinônimo de Veículos Autônomos Subaquáticos (AUVs, do inglês *Autonomous Underwater Vehicle*).

categoria UUV ainda pode ser dividida em outras duas, os Veículos Autônomos Subaquáticos (AUVs, do inglês *Autonomous Underwater Vehicle*) e os Veículos Remotamente Operados (ROV, do inglês *Remoted Operated Vehicle*). A principal característica que diferencia um ROV de um AUV é a presença de cabos que são responsáveis por realizar a comunicação direta do robô com a superfície. No caso dos ROVs, necessita-se de uma série de cabos que o liguem à plataforma para que o operador controle o robô, transmitindo comandos para o ROV (Christ; Wernli, 2018). Na Figura 3 está uma versão traduzida do fluxograma detalhado na referência.

Figura 3 – Classificação dos veículos subaquáticos



Fonte: adaptado de Christ e Wernli (2018).

Pode-se, então, genericamente reconhecer um ROV pelo fato de ter em seu corpo uma câmera dentro de um invólucro à prova de água, acompanhado com motores que possibilitam sua movimentação, além do cabeamento para controle.

### 3.3 PROJETOS QUE UTILIZAM ROVS

Para maior compreensão e contextualização do tema e trabalho desenvolvido, foram analisados alguns ROVs e projetos, os quais serão expostos nos próximos subcapítulos.

#### 3.3.1 Projeto ARGO

O projeto ARGO permite o monitoramento detalhado e de longo prazo de mudanças climáticas nos oceanos, como o aquecimento e salinidade das águas. Ele dispõe de uma frota de aproximadamente 4000 flutuadores monitorando os oceanos. Uma imagem da distribuição dos flutuadores é vista na Figura 4, mostrando a sua abrangência global.

Figura 4 – Imagem da constelação de veículos do projeto ARGO



Fonte: Fleetmonitoring (2023).

### 3.3.2 ROV Jason, da WHOI

O ROV Jason faz parte dos equipamentos da organização Woods Hole Oceanographic Institution (WHOI) e pertence a um dos diversos setores do Laboratório de Submersão Profunda (conhecido como DSL, do inglês “*Deep Submergence Laboratory*”), laboratório que o construiu com a arrecadação de recursos cedidos pelo National Science Foundation. O veículo permite que cientistas tenham acesso ao fundo mar enquanto realizam suas pesquisas do deck de um navio.

Um cabo de fibra ótica reforçado de 10 quilômetros provê a força e os comandos para Jason, que, por sua vez, retorna dados e imagens de vídeo ao operador (ROV, 2023). Durante uma de suas expedições em 2009, o veículo Jason registrou, observou e coletou amostras de uma erupção no vulcão submarino West Mata, localizado a 200 km das ilhas Samoa e ocorrida a 1200 m de profundidade, conforme pode ser observado na Figura 5.

Figura 5 – Imagem do ROV Jason na erupção do West Mata



Fonte: Piecuch (2023).

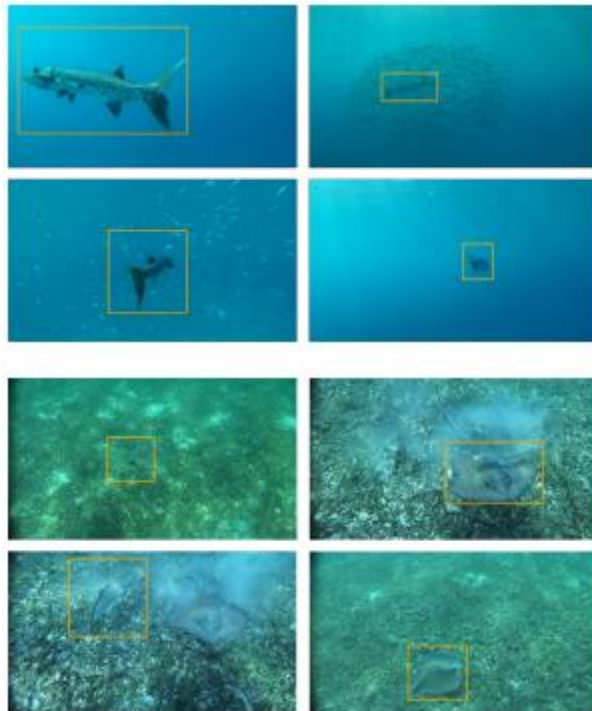
### 3.3.3 Projeto CUREE

Os recifes de corais são sensíveis aos distúrbios causados pelos seres humanos e mudanças climáticas. Logo, para a preservação desses ecossistemas,

destaca-se a importância de ferramentas que possibilitem o monitoramento, compreensão, avaliação e mensurações das intervenções e alterações ao longo do tempo.

O projeto *Curious Robot for Ecosystem Exploration* (CUREE, em tradução livre para o português Robô Curioso para Exploração de Ecossistema) propõe a criação de um sistema robótico autônomo que pode explorar estes ecossistemas subaquáticos, observar interações complexas entre organismos e habitats e adaptar seu comportamento em tempo real para auxiliar nos estudos da fauna marinha. Na Figura 6 é possível ver o veículo identificando algumas espécies.

Figura 6 – Imagens do projeto CUREE identificando espécies



Fonte: Girdhar (2023).

### 3.3.4 BlueROV2

O BlueROV2 (Figura 7) é disponibilizado pela empresa BlueRobotics em um kit parcialmente montado. Em até 8 horas o usuário consegue realizar sua montagem e configuração, além de aprender como realizar modificações futuras e reparos quando necessários.

Um aspecto a se destacar é que ele é um ROV com classe de observação, logo o usuário pode utilizá-lo para inspeções simples ou então realizar adaptações no equipamento com a instalação de acessórios como manipuladores.

O BlueROV2 utiliza a plataforma Raspberry Pi 4 para processamento e controle, já para a interface dos sensores e motores ele utiliza a placa Navigator, a qual é descrita na seção 7.

Uma distribuição de sistema operacional própria da BlueRobotics, chamada BlueOS, roda neste ROV. O BlueOS, por sua vez, é baseado no sistema operacional oficial da Raspberry Pi Foundation, o Raspbian.

Figura 7 – BlueROV2 da BlueRobotics em configuração padrão



Fonte: BlueRobotics (2023).

## 4 LINGUAGEM RUST

Algumas linguagens como Go (Google), Swift (Apple), .NET (Microsoft) possuem certos riscos inerentes à governança das próprias empresas que as detêm. Estas empresas podem adotar medidas que venham a favorecer seus produtos (Matthews, 2022), o que pode criar impedimentos e problemas para desenvolvedores que optem por usar estas linguagens em seus projetos.

Uma alternativa mais flexível é a linguagem Rust, que é um projeto *open-source* e dirigido por uma comunidade, inicialmente gerida de forma não comercial pela Mozilla Foundation. O Rust é fornecido por meio de duas licenças, a Apache e a MIT.

Deste modo, projetos individuais que usam o Rust conseguem adicionar seus próprios termos em licenças separadas. Por esse motivo, observa-se uma forte aderência de empresas de tecnologia à linguagem Rust, como a Amazon, Facebook, Google, Apple, Microsoft e outros. Isso, porque a linguagem não está associada a uma companhia específica, então é uma boa escolha a longo prazo, com mínimos conflitos de interesse econômico.

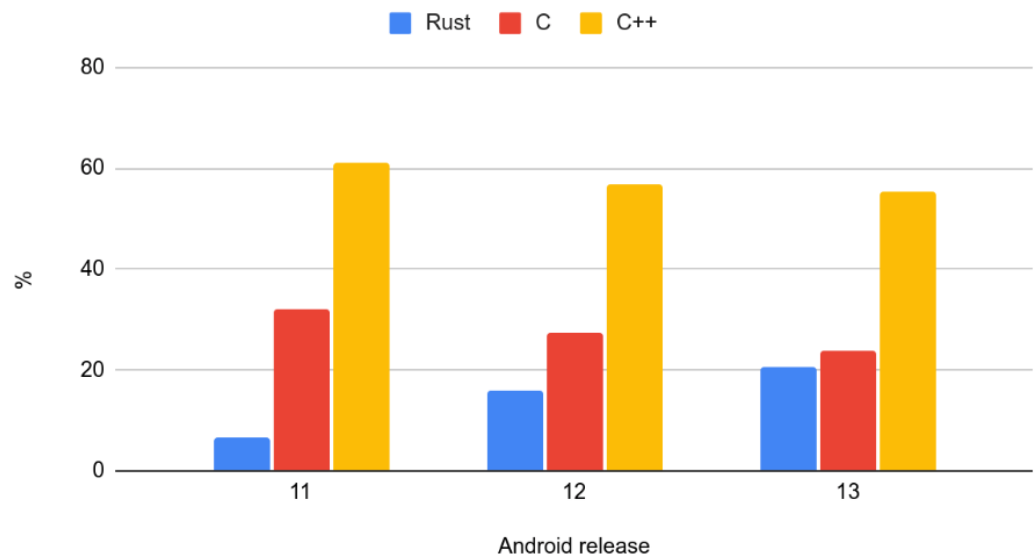
### 4.1 RUST NO PROJETO ANDROID

Em uma publicação da Google, referente ao projeto Android, de 2019 a 2022, o número anual de vulnerabilidades de segurança de memória caiu de 223 para 85. Esse declínio coincide com uma política adotada pela empresa de buscar maior uso de linguagens de programação seguras para a memória. Inclusive o Android 13 é a primeira versão em que a maioria do novo código adicionado é escrito em uma linguagem segura para a memória (Stoep, 2022).

Na Figura 8 pode-se notar um aumento no uso da linguagem Rust no Android, observado que na versão 13 tem-se cerca de 21% de todo o novo código nativo escrito em Rust. Até o momento, não foi reportada nenhuma vulnerabilidade de segurança de memória nos códigos Rust, o que demonstra seu valor na prevenção de vulnerabilidades (Stoep, 2022).



Figura 8 – Gráfico destacando a progressão da linguagem Rust no Android  
New Native Code



Fonte: Stoep (2023).

Em 2023, a empresa continua a desenvolver e utilizar a linguagem, que pode ser encontrada nas recentes atualizações do aparelho Google Pixel 6, por exemplo (Walbran, 2023).

## 4.2 RUST NA MICROSOFT

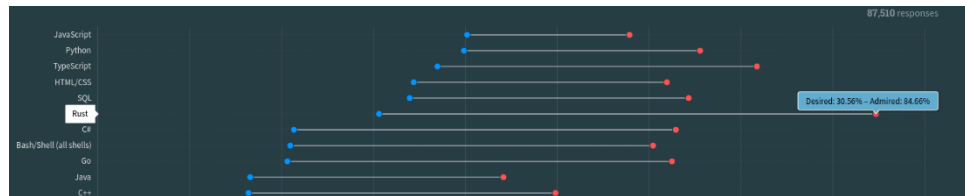
A Microsoft também está aplicando a linguagem Rust, como mencionado pelo CTO da Azure, Mark Russinovich, em uma versão recente do Windows 11 Insider Preview. A mudança visa combater vulnerabilidades e explorações, com a memória segura sendo uma prioridade. A Microsoft começou a adotar Rust após anunciar seus planos em abril de 2023 (Langowski; Leblanc, 2023).

## 4.3 STACK OVERFLOW

O StackOverflow, fórum comum de utilização por desenvolvedores, também destaca a linguagem Rust como uma das mais desejadas e preferidas aos desenvolvedores participantes de uma pesquisa anual (DEVELOPER, 2023). A

linguagem se destaca como preferida por mais de 80% dos desenvolvedores, como visto na Figura 9.

Figura 9 – Preferência de Linguagens de Programação na Pesquisa de Desenvolvedores do Stack Overflow



Fonte: Developer (2023).

## 5 DESENVOLVIMENTO – RUST

O projeto Rust disponibiliza um repositório de bibliotecas online no endereço crates.io, no qual empresas e hobbistas podem compartilhar suas bibliotecas e documentações com o público

O desenvolvimento das bibliotecas para a plataforma Navigator baseou-se em bibliotecas já disponíveis no crates.io, que foram testadas e adaptadas aos propósitos deste trabalho. Nos próximos subcapítulos são elencados alguns conceitos e pesquisas referentes ao desenvolvimento destas bibliotecas. Por fim, detalhes do processo de desenvolvimento são descritos nos próximos capítulos.

### 5.1 RUST – CROSS-COMPILING

Os binários gerados pela linguagem Rust podem ser compilados para diferentes plataformas. O programa é compatível com uma variedade de sistemas operacionais, como Windows, MacOS e Linux, abrangendo também diferentes arquiteturas de processadores.

Entre as plataformas disponíveis, temos as baseadas na arquitetura ARM, armv7 e aarch64 (também é conhecida como armv8). Tais arquiteturas são utilizadas atualmente pela Raspberry Pi 4 em seus produtos, além de seu sistema operacional oficial, o Raspbian.

O Raspbian utiliza da biblioteca “*libc*” sob licença GNU. Para criar suas próprias soluções sem a necessidade de expor seus códigos a terceiros, o usuário pode utilizar o musl, uma alternativa que fornece o “*libc*” com licença menos restritiva. O Rust também disponibiliza compilação utilizando o “*libc*” provido pela musl, para plataformas armv7 e aarch64.

Sendo assim, neste projeto a biblioteca será disponibilizada e testada para quatro opções de arquitetura, ou targets, como chamados no ambiente de Rust (PLATFORM, 2023), como:

- armv7-unknown-linux-gnueabihf.
- aarch64-unknown-linux-gnu.

- armv7-unknown-linux-musleabihf.
- aarch64-unknown-linux-musl.

## 5.2 RUST – FFI

Ocasionalmente, é possível que o programador precise de conexões para usar um código em outra linguagem, ou até exportá-lo de forma compatível, como é a necessidade e parte da exploração deste trabalho. Este recurso é conhecido como Interface de função externa (FFI, do inglês *Foreign Function Interface*). Por meio do projeto *bindgen*, o usuário consegue utilizar estruturas comuns à linguagem C, o que possibilita sua intercambialidade.

Com o recurso de FFI disponível em algumas linguagens, o Rust pode fornecer trechos de códigos aproveitando de seu desempenho e segurança, o que é muito comum em servidores, por exemplo.

Os recursos apresentados são desejáveis neste trabalho, visto que atendem ao requisito de conexão com a linguagem C++ e com o Python.

### 5.2.1 FFI e C++

Por meio do *bindgen*, o programador pode gerar biblioteca compatível com a linguagem C e então escrever manualmente um arquivo *header*, que identificará as funções. O projeto *cbindgen* possibilita ao usuário gerar o *header* de forma automatizada (Mozilla, *cbindgen*).

Com a integração das ferramentas mencionadas, é possível gerar o arquivo binário da biblioteca com suas funções mapeadas pelo *header*.

Com os dois arquivos mencionados, o código escrito em C++ e um compilador, como o Cmake, o programador poderá compilar seu código.

### 5.2.2 FFI e Python

A conexão entre linguagens também é disponível para Python, mas alguns projetos facilitam a integração e geração de bibliotecas. Neste trabalho, as tecnologias estudadas para a aplicação foram o `pyo3` (GitHub 2023a) para realizar os *bindings*, e o *maturin* (GitHub, 2023b) para gerar o pacote a ser distribuído para o Python.

O projeto Maturin permite criar *wheels*, um formato de instalação popular para linguagem Python por meio do Pip. O Pip é um software que promove conexão com uma vasta biblioteca de códigos compartilhada, conhecida como *Python Packages Integration* (PyPI).

O Python se encontra disponível em versões de 3.7 a 3.13. Na Tabela 1, retirada do site oficial, podem ser verificadas as datas de lançamento e período de suporte para cada versão. As versões novas de Python geralmente visam introduzir uma série de aprimoramentos na linguagem. Por exemplo, incluindo: novos operadores e funcionalidades para tornar o código mais conciso e legível, adições na biblioteca padrão, melhorias na tipagem e verificação de tipos, otimizações de desempenho e novos recursos.

Tabela 1 – Tabela das versões disponíveis Python

Versão	Estado de manutenção	Primeiro lançamento	Final do suporte	Agendamento
3.13	prerelease	2024-10-01 (planejado)	2029-10	PEP 719
3.12	bugfix	02/10/2023	2028-10	PEP 693
3.11	bugfix	24/10/2022	2027-10	PEP 664
3.10	security	04/10/2021	2026-10	PEP 619
3.9	security	05/10/2020	2025-10	PEP 596
3.8	security	14/10/2019	2024-10	PEP 569

Fonte: Adaptado de Active, (2023).

Como padrão, cada biblioteca de Python deve possuir uma compilação para cada versão. Para integrar as bibliotecas de diferentes versões, a organização lançou uma padronização, denominada PEP 384 (Von Löwis, 2009). Essa padronização introduziu o conceito de manter o uso de uma API simplificada com funções limitadas,

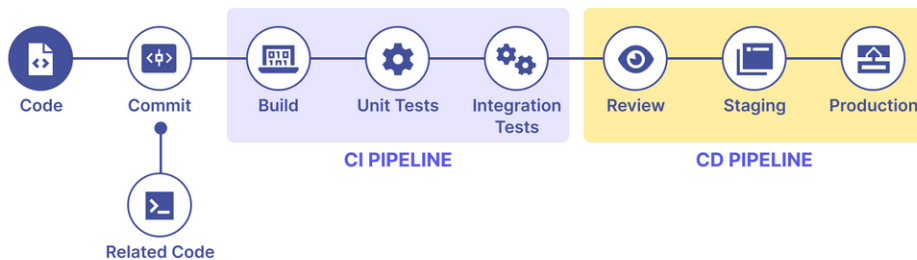
conhecida também como `abi3`, uma especificação que permite uma única biblioteca rodar em diferentes versões de Python.

## 6 DESENVOLVIMENTO - SISTEMA DE VERSIONAMENTO E CI/CD

O GitHub é a ferramenta para gerenciamento de repositórios padrão na empresa BlueRobotics. O Git é um software que possibilita versionar o código e ter controle do fluxo de trabalho em pacotes definidos. A plataforma GitHub fornece um ambiente para gerenciar os projetos, repositórios e promover discussões acerca dos trabalhos propostos.

Duas práticas comuns na cultura ágil de desenvolvimento de software são Integração Contínua e Entrega Contínua, ou o CI/CD. O GitHub disponibiliza a ferramenta GitHub Actions que possibilita realizar tais práticas (Git Hub Docs, 2023). Na Figura 10 pode-se ver como é dado o fluxo de trabalho do código versionado e integrado ao CI/CD, na figura pode-se observar como se dá a integração de um pacote de modificações aplicada ao projeto.

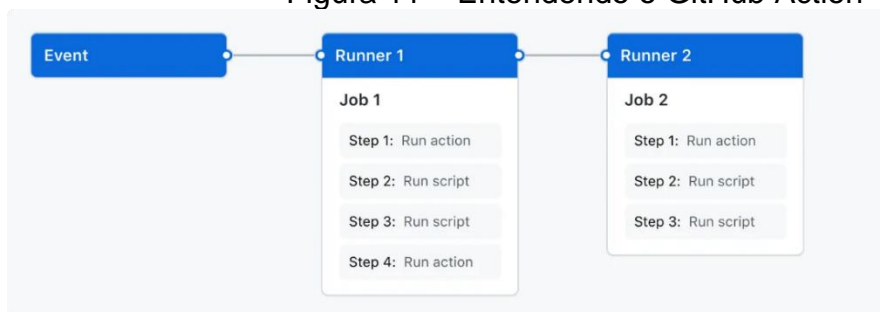
Figura 10 – Fluxo de trabalho no desenvolvimento



Fonte: GitHub Docs (2023).

O GitHub Actions funciona por meio de fluxos de trabalho. Por meio de um arquivo em extensão *yaml*, o programador define os eventos, ações, trabalhos e executores. A Figura 11 apresenta o fluxo de execução.

Figura 11 – Entendendo o GitHub Action



Fonte: GitHub Docs (2023).

Nos eventos tem-se a inicialização de um fluxo de trabalho, que pode ser por um trecho de código atualizado, um pedido de revisão entre outros motivos.

Realizado o início do fluxo de trabalho, os trabalhos do arquivo serão executados. Uma característica interessante é que eles podem ser estruturados para executar em paralelo ou sequenciais.

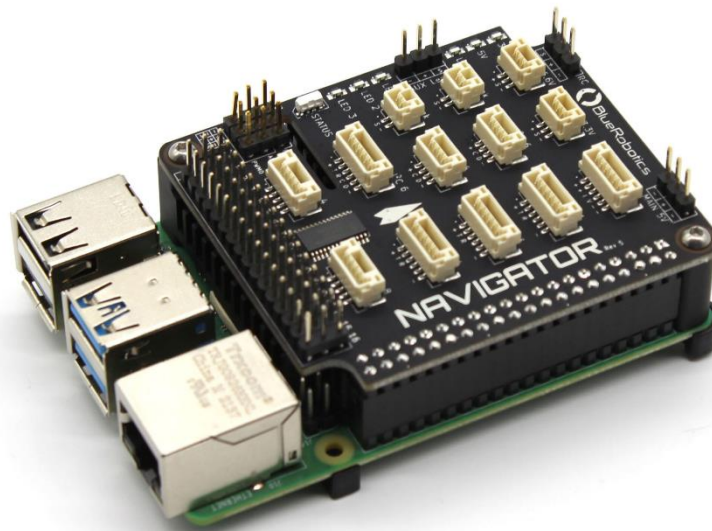
Os trabalhos são executados por servidores disponibilizados pelo GitHub. Durante os avanços no desenvolvimento, descobriu-se o uso do executor *self-hosted*. Por meio de um script, a organização detentora dos repositórios no GitHub pode conectar-se a máquinas locais para executar os códigos do CI/CD. Tal prática foi utilizada com o Raspberry Pi 4 e será abordada nas categorias de desenvolvimento.



## 7 DESENVOLVIMENTO - PROJETO NAVIGATOR-RS

O Navigator (Figura 12) é um acessório para a plataforma Raspberry Pi 4. Ele tem um projeto pensado na sua compatibilidade, conhecido como *hardware attached top* (HAT), um acessório que pode ser conectado às GPIO do microcomputador e ampliar suas funcionalidades (Adams, 2014).

Figura 12 – A placa Navigator conectada a um Raspberry Pi 4



Fonte: BlueRobotics (2023).

O *hardware* Navigator é equipado com uma diversidade de sensores para a navegabilidade do ROV, como acelerômetro, giroscópio, magnetômetro, barômetro, termômetro digital, leds de status e Modulação por Lagura de Pulso (PWM, do inglês *Pulse Width Modulation*) para controle dos motores.

A comunicação com os sensores disponíveis no hardware requer a configuração do Raspberry Pi 4 para utilizar suas portas GPIO como entradas e saídas digitais, além de configurá-las para comunicação com barramentos I2C e SPI. Parte do processo será explanada no capítulo seguinte.

## 7.1 ESTUDO E INTEGRAÇÃO DO HARDWARE

A fim de propor e realizar o trabalho de integração dos sensores em uma biblioteca, fez-se necessário um levantamento detalhado dos sensores disponíveis na placa, assim como suas configurações realizadas no *layout* físico da placa, uma vez que estas definem os pinos de seleção e endereços. Na Tabela 2 são listadas as configurações levantadas para utilização da Navigator.

Tabela 2 – Tabela listando os componentes da Navigator

Ref.	Sensor	Descrição	Dispositivo	Pinos do Raspberry	Endereço
U1	PCA9685PW	16-channel PWM	i2c-4	SDA4 : GPIO6, SCL4 : GPIO7, (ALT5)	0x40
U2	BMP280	3-in-1 sensor, humidity, pressure, temperature	i2c-1	SDA1 : GPIO2, SCL1 : GPIO3	0x76
U3	M24C32-DR	EEPROM	i2c-0	SDA0 : GPIO0, SCL0 : GPIO1	0x50
U4	ADS1115IDGS	16-Bit ADC	i2c-1	SDA1 : GPIO2, SCL1 : GPIO3	0x48
U5	ICM-20602	IMU – 6-Axis MEMS MotionTracking	SPI-1	MISO1 : GPIO19, MOSI1 : GPIO20, SCLK1 : GPIO21, CS2 : GPIO16, (ALT4)	CS2
U15	MMC5983	3-axis Magnetic Sensor	SPI-1	MISO1 : GPIO19, MOSI1 : GPIO20, SCLK1 : GPIO21, CS1 : GPIO17, (ALT4)	CS1
U16	AK09915	3-axis Electronic Compass	i2c-1	SDA1 : GPIO2, SCL1 : GPIO3	0x0C
U17	SK6812-SIDE-A	Integrated LED	unipolar RZ	RGB_LED : GPIO10	
D5	STATUS_LED1			GPIO24	
D6	STATUS_LED2			GPIO25	
D7	STATUS_LED3			GPIO11	

Fonte: elaborado pelo autor.

Com o levantamento das configurações necessárias, pôde-se avaliar a configuração da Raspberry Pi 4 por meio de *overlays*, configurações alternativas que integram funcionalidades às GPIOs, tais configurações foram realizadas utilizando a Tabela 3 fornecida pelo fabricante.

**Tabela 3 – Funções alternativas por GPIO no Raspberry Pi 4**

GPIO	Default						
	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
0	High	SDA0	SA5	PCLK	SPI3_CE0_N	TXD2	SDA6
1	High	SCL0	SA4	DE	SPI3_MISO	RXD2	SCL6
2	High	SDA1	SA3	LCD_VSYNC	SPI3_MOSI	CTS2	SDA3
3	High	SCL1	SA2	LCD_HSYNC	SPI3_SCLK	RTS2	SCL3
4	High	GPCLK0	SA1	DPI_D0	SPI4_CE0_N	TXD3	SDA3
5	High	GPCLK1	SA0	DPI_D1	SPI4_MISO	RXD3	SCL3
6	High	GPCLK2	SOE_N	DPI_D2	SPI4_MOSI	CTS3	SDA4
7	High	SPI0_CE1_N	SWE_N	DPI_D3	SPI4_SCLK	RTS3	SCL4
8	High	SPI0_CE0_N	SD0	DPI_D4	-	TXD4	SDA4
9	Low	SPI0_MISO	SD1	DPI_D5	-	RXD4	SCL4
10	Low	SPI0_MOSI	SD2	DPI_D6	-	CTS4	SDA5
11	Low	SPI0_SCLK	SD3	DPI_D7	-	RTS4	SCL5
12	Low	PWM0	SD4	DPI_D8	SPI5_CE0_N	TXD5	SDA5
13	Low	PWM1	SD5	DPI_D9	SPI5_MISO	RXD5	SCL5
14	Low	TXD0	SD6	DPI_D10	SPI5_MOSI	CTS5	TXD1
15	Low	RXD0	SD7	DPI_D11	SPI5_SCLK	RTS5	RXD1
16	Low	FL0	SD8	DPI_D12	CTS0	SPI1_CE2_N	CTS1
17	Low	FL1	SD9	DPI_D13	RTS0	SPI1_CE1_N	RTS1
18	Low	PCM_CLK	SD10	DPI_D14	SPI6_CE0_N	SPI1_CE0_N	PWM0
19	Low	PCM_FS	SD11	DPI_D15	SPI6_MISO	SPI1_MISO	PWM1
20	Low	PCM_DIN	SD12	DPI_D16	SPI6_MOSI	SPI1_MOSI	GPCLK0
21	Low	PCM_DOUT	SD13	DPI_D17	SPI6_SCLK	SPI1_SCLK	GPCLK1
22	Low	SD0_CLK	SD14	DPI_D18	SD1_CLK	ARM_TRST	SDA6
23	Low	SD0_CMD	SD15	DPI_D19	SD1_CMD	ARM_RTCK	SCL6
24	Low	SD0_DAT0	SD16	DPI_D20	SD1_DAT0	ARM_TDO	SPI3_CE1_N
25	Low	SD0_DAT1	SD17	DPI_D21	SD1_DAT1	ARM_TCK	SPI4_CE1_N
26	Low	SD0_DAT2	TE0	DPI_D22	SD1_DAT2	ARM_TDI	SPI5_CE1_N
27	Low	SD0_DAT3	TE1	DPI_D23	SD1_DAT3	ARM_TMS	SPI6_CE1_N

Fonte: Raspberry Pi 4 Model B (2019).

A configuração final é gerada por uma lista de *overlays* que será executada na inicialização do sistema operacional, preparando o Raspberry Pi 4 para disponibilizar as portas requisitadas.

Além da configuração do Raspberry Pi 4 para se comunicar com os periféricos, é essencial o uso de bibliotecas para intermediar a interação entre o usuário e a manipulação dos registradores nesses sensores. Os próximos subtópicos irão descrever o processo e aplicação do código já integrado à biblioteca.

### 7.1.1 O Modulo PWM

O controle do Modulação por Largura de Pulso (PWM, do inglês *Pulse Width Modulation*) é realizado pelo Circuito Integrado (CI) PCA9685, um controlador para LEDs e servomotores com 16 canais comandados de forma independente.

O CI suporta uma frequência ajustável, que também depende do seu oscilador. No *layout* da Navigator, há um cristal externo de 24,5760 MHz anexado para melhorar sua precisão. Então, na biblioteca, o oscilador externo deve ser ativado.

Para realizar a mudança de frequência no CI, deve-se alterar o registrador PRE\_SCALE, e este valor deve ser calculado conforme anunciado na especificação do fabricante, pela equação (1).

$$prescale_{value} = \text{round} \left( \frac{f_{clock}}{(4096 \times f_{desired})} \right) - 1 \quad (1)$$

Pela descrição na documentação fornecida pelo fabricante, o menor valor do *prescaler* deve ser 3, que corresponde a 1526 Hz e seu máximo 255, com uma frequência de 24 Hz.

Logo, se no exemplo da Navigator constar como ajustar a frequência para 60 Hz, basta ajustar o *prescaler* para 100. Internamente, caso o oscilador já estiver em uso, esse procedimento interrompe-o e reinicia-o.

O pino *Output Enable* (OE, em tradução livre para o português, Habilidade de Saída), o qual é ligado a uma GPIO do Raspberry, que pode realizar a ativação/desativação do CI de forma digital.

Para realizar o controle da largura de pulso, alteram-se os contadores no registro, um referente a ON e outro referente a OFF. Simplificando o uso, quando ON é fixado em 0, o valor de OFF determina o Duty Cycle do canal selecionado.

No BlueROV2, o PWM tem a função principal de comandar os propulsores (Figura 13) e o sistema de iluminação (Figura 14 e Figura 15). No Código 1 tem-se um exemplo da biblioteca sendo utilizada após as integrações e configurações necessárias.

Figura 13 – Thruster T200 da BlueRobotics



Fonte: BlueRobotics (2023).

Figura 14 – Lumen Subsea



Fonte: BlueRobotics (2023)

Figura 15 – Lumen Light em uso



Lights off

Lumen Brightness Range

Fonte: BlueRobotics (2023).

## Código 1 – Exemplo da biblioteca utilizando modulo PWM

Further info

Check 7.3.5 - **PWM** frequency PRE\_SCALE

Examples

```

1 use navigator_rs::{Navigator, PwmChannel};
2
3 let mut nav = Navigator::new();
4
5 nav.init();
6 nav.pwm_enable(true);
7
8 nav.set_pwm_freq_prescale(99); // sets the pwm frequency to 60 Hz
9
10 nav.set_pwm_channel_value(PwmChannel::Ch1, 2048); // sets the duty cycle to 50%

```

Fonte: elaborado pelo autor.

### 7.1.2 Acelerômetro E Giroscópio

O sensor ICM20689 é compatível com as interfaces I2C e SPI. Na configuração da Navigator, é feito uso do barramento SPI, exigindo uma GPIO para a seleção do canal SPI.

O IC ICM20689 é um sensor de 6 graus de liberdade (6-DoF), o que significa que ele é capaz de medir aceleração linear em três eixos e a taxa de rotação angular em três eixos.

As medidas da aceleração do sensor são fornecidas em termos da constante gravitacional. Realizando uma aproximação, é possível ter o valor da aceleração em  $m/s^2$  para os três eixos. O giroscópio fornece suas medidas em rad/s.

Essas medições fornecem ao ROV um sistema de monitoramento inercial, que pode ser realizado pelo processamento das leituras dos sensores. Além disso, esse sensor pode usar o magnetômetro, adicionando mais uma variável às medidas de posicionamento do ROV.

Um exemplo do uso é apresentado no Código 2 e Código 3.

## Código 2 – Exemplo da biblioteca utilizando modulo ICM20689

```
[-] pub fn read_accel(&mut self) -> AxisData source
Reads acceleration based on ICM20689 of Navigator.
Measurements in [m/s2]
Examples
1 use navigator_rs::{Navigator};
2 use std::thread::sleep;
3 use std::time::Duration;
4
5 let mut nav = Navigator::new();
6 nav.init();
7
8 loop {
9     let accel = nav.read_accel();
10    println!("accel values: X={}, Y={}, Z={} [m/s2]", accel.x, accel.y, accel.z);
11    sleep(Duration::from_millis(1000));
12 }
```

Fonte: elaborado pelo autor.

## Código 3 – Exemplo da biblioteca utilizando modulo ICM20689 b

```
[-] pub fn read_gyro(&mut self) -> AxisData source
Reads angular velocity based on ICM20689 of Navigator.
Measurements in [rad/s]
Examples
1 use navigator_rs::{Navigator};
2 use std::thread::sleep;
3 use std::time::Duration;
4
5 let mut nav = Navigator::new();
6 nav.init();
7
8 loop {
9     let gyro = nav.read_gyro();
10    println!("accel values: X={}, Y={}, Z={} [rad/s]", gyro.x, gyro.y, gyro.z);
11    sleep(Duration::from_millis(1000));
12 }
```

Fonte: elaborado pelo autor.

### 7.1.3 Conversor Analógico Digital (ADC)

A placa Navigator dispõe de um Conversor Analógico-Digital (ADC, do inglês *analog-to-digital converter*) de quatro canais, o ADS1115, interfaceado por meio do barramento I2C.

O conversor possui quatro canais, mas apenas um deles pode ser aferido por leitura realizada, um multiplexador realiza a seleção do canal. A leitura possui a resolução de 16 bits e um ganho ajustável, permitindo que o conversor se adeque às faixas de leitura. Também é possível ajustar a taxa de aquisição do sensor, embora maiores taxas tendam a ter maior ruído associado.

Na Navigator, o ADC é disponível para medições externas nos canais 1 e 2. Os canais 3 e 4 ficam reservados à medição do nível de carga da bateria do ROV. No Código 4 temos um exemplo de sua utilização após integrado à biblioteca.

#### Código 4 – Exemplo da biblioteca utilizando modulo ADS1115

```
[ ] pub fn read_adc(&mut self, channel: AdcChannel) -> f32 source
    Reads the ADC based on ADS1115 of Navigator.
    Measurements in [V]
    Examples
    1 use navigator_rs::{AdcChannel, Navigator};
    2 use std::thread::sleep;
    3 use std::time::Duration;
    4
    5 let mut nav = Navigator::new();
    6 nav.init();
    7
    8 loop {
    9     println!("ADC channel 1 value: {} [V]", nav.read_adc(AdcChannel::Ch1));
    10    sleep(Duration::from_millis(1000));
    11 }
```

Fonte: elaborado pelo autor.

### 7.1.4 Magnetômetro

O magnetômetro AK09915 não possuía uma biblioteca para sua utilização em Rust. Para a realização deste trabalho, desenvolveu-se uma biblioteca, disponível na plataforma crates.io, sob o título de ak09915\_rs.

Nesta biblioteca foi possível empregar o projeto `embedded_hal`. A sigla HAL se refere à Camada de Abstração de Hardware (do inglês, *Hardware Abstraction Layer*). Uma vez que se programou uma biblioteca usando HAL, pode-se incorporá-la a diferentes plataformas, bastando integrar seus componentes. Além de utilizar este sensor na Raspberry Pi 4, é possível usá-lo em um microcontrolador. Outro benefício é um maior número de usuários para testar e melhorar a biblioteca conforme os projetos aumentem.

Uma parte em destaque para este sensor é a disponibilidade da função de *self-test*. De acordo com a especificação do fabricante, uma rotina de teste interno é acionada ao selecionar o modo *self-test*. Após a leitura estar pronta, o bit `Data_Read` está em nível alto, permitindo a comparação dos valores medidos.



Outra possibilidade explorada no sensor foi o modo de leitura contínuo. Nele, um período de leitura é ajustado ao padrão de 200 Hz. A leitura sempre é indicada pelo bit *data\_read* e é retida até que o *overflow\_check* seja verificado.

A leitura é realizada nos registradores respectivos para os 3 eixos, tendo 2 bytes cada. O valor deve ser multiplicado pela escala do fabricante, de 0,15, a sensibilidade (BSE) para temperatura ambiente de 25 °C [ $\mu\text{T}/\text{LSB}$ ]. Um exemplo da utilização do sensor integrado a biblioteca está no Código 5.

### Código 5 – Exemplo da biblioteca utilizando modulo AK09915

```
pub fn read_mag(&mut self) -> AxisData source
    Reads the magnetometer Ak09915 of Navigator.
    Measurements in [ $\mu\text{T}$ ]
    Examples
1  use navigator_rs::{Navigator};
2  use std::thread::sleep;
3  use std::time::Duration;
4
5  let mut nav = Navigator::new();
6  nav.init();
7
8  loop {
9      let mag = nav.read_mag();
10     println!("mag values: X={}, Y={}, Z={} [ $\mu\text{T}$ ]", mag.x, mag.y, mag.z);
11     sleep(Duration::from_millis(1000));
12 }
```

Fonte: elaborado pelo autor.

#### 7.1.5 Sensor de temperatura e pressão

O sensor BMP280 tem interfaces I2C e SPI. Na estrutura da Navigator, ele é conectado via SPI, o que requer a utilização de um pino GPIO para selecionar o canal SPI.

Com o sensor, é possível medir a temperatura interna do ambiente no local em que a placa está localizada. Em uma aplicação prática, isso poderia ser no invólucro do submarino, como mostrado na Figura 16.

O sensor de pressão também pode ser utilizado para estimar a altitude do veículo por meio da pressão. Um exemplo do uso do sensor é disponibilizado no Código 6 e Código 7.

Figura 16 – Cápsulas submarinas



Fonte: BlueRobotics (2023)

Código 6 – Exemplo da biblioteca utilizando módulo BMP280

```
[~] pub fn read_pressure(&mut self) -> f32 source
Reads the pressure based on BMP280 of Navigator.
Measurements in [kPa]
Examples
1 use navigator_rs::{Navigator};
2 use std::thread::sleep;
3 use std::time::Duration;
4
5 let mut nav = Navigator::new();
6 nav.init();
7
8 loop {
9     let pressure = nav.read_pressure();
10    println!("pressure value: {pressure} [kPa]");
11    sleep(Duration::from_millis(1000));
12 }
```

Fonte: elaborado pelo autor.

Código 7 – Exemplo da biblioteca utilizando modulo BMP280

```
[~] pub fn read_temperature(&mut self) -> f32 source
Reads the temperature using BMP280 of Navigator.
Measurements in [°C]
Examples
1 use navigator_rs::{Navigator};
2 use std::thread::sleep;
3 use std::time::Duration;
4
5 let mut nav = Navigator::new();
6 nav.init();
7
8 loop {
9     let temperature = nav.read_temperature();
10    println!("temperature value: {temperature} [°C]");
11    sleep(Duration::from_millis(1000));
12 }
```

Fonte: elaborado pelo autor.

### 7.1.6 LEDs de Status e Neopixel

Na placa se encontram disponíveis 3 LEDs nas cores vermelho, verde e azul. Estes são interfaceados com a placa por meio das GPIOs. A Navigator também dispõe de um led RGB, ou neopixel, com uma gama de cores para servir como indicadores.

Um exemplo disponível no repositório é a detecção de inclinação. No código, tem-se um *loop* que realiza a medida do acelerômetro e, comparando a medição obtida a um valor predefinido, aciona, ou não, um LED indicativo.

O LED neopixel também dispõe de um barramento para expansão, permitindo concatenar outros LED RGB.

No Código 8 tem-se um exemplo da utilização da biblioteca para controlar os LEDs, no Código 9 um exemplo do controle do neopixel.

#### Código 8 – Exemplo da biblioteca utilizando os LEDs

```
[H] pub fn set_led(&mut self, select: UserLed, state: bool) source
```

Sets the selected navigator LED output.

**Arguments**

- `select` - A pin selected from `UserLed`.
- `state` - The value of output, LED is on with a true logic value.

**Examples**

```

1 use navigator_rs::{UserLed, Navigator};
2 use std::thread::sleep;
3 use std::time::Duration;
4
5 let mut nav = Navigator::new();
6
7 nav.init();
8 loop {
9     nav.set_led(UserLed::Led1, true);
10    sleep(Duration::from_millis(1000));
11    nav.set_led(UserLed::Led1, false);
12    sleep(Duration::from_millis(1000));
13 }
```

Fonte: elaborado pelo autor.

### Código 9 – Exemplo da biblioteca utilizando o neopixel

```
pub fn set_neopixel(&mut self, array: &[[u8; 3]])
```

Set the values of the neopixel LED array.

**Arguments**

- `array` - A 2D array containing RGB values for each LED. Each inner array is a [u8; 3] representing the Red, Green and Blue from a LED.

**Example**

```
1 use navigator_rs::{Navigator};
2
3 let mut nav = Navigator::new();
4
5 nav.init();
6 let mut leds = [[0, 0, 255], [0, 255, 0], [255, 0, 0]];
7 nav.set_neopixel(&mut leds);
```

This will set the first LED to blue, second to green, and third to red.

Fonte: elaborado pelo autor.

## 7.2 INTEGRAÇÃO

Como visto em exemplos nos códigos anteriores, a biblioteca criada contém o objeto Navigator. O Navigator dispõe em sua estrutura os módulos: adc, bmp, pwm, mag, imu, led e neopixel, que foram apresentados nos subcapítulos anteriores. Esta estrutura é mostrada no Código 10.

### Código 10 – Código com estrutura do objeto Navigator

```
pub struct Navigator {
    pwm: Pwm,
    bmp: Bmp280,
    adc: Ads1x1x<I2cInterface<I2cdev>, Ads1115, Resolution16Bit,
ads1x1x::mode::OneShot>,
    imu: ICM20689<SpiInterface<Spidev, Pin>>,
    mag: Ak09915<I2cdev>,
    led: Led,
    neopixel: Strip,
}
```

Fonte: elaborado pelo autor.

O programador, por sua vez, pode utilizar o método `new()`, como visto em alguns exemplos, o qual criará os objetos e irá atribuí-los a estrutura Navigator.

O Código 11 é disponibilizado junto à biblioteca no repositório. Neste código, tem-se um demonstrativo simples do sensor criando as dependências necessárias pelo método *new*.

O método *init* realiza a pré-configuração de todos os sensores e periféricos para uma configuração padrão.

Por fim, a leitura dos sensores pode ser realizada de forma simples por um método *fmt\_debug*, como demonstrado no código, ou individualmente como apresentado nos códigos anteriores.

A execução do código mostrará ao usuário via terminal a leitura dos sensores, como pode ser visto na Figura 17.

#### Código 11 – Código exemplo

```
use navigator_rs::{Navigator, PwmChannel};
use std::thread::sleep;
use std::time::Duration;
fn main() {
    println!("Creating your navigator module!");
    let mut nav = Navigator::new();

    println!("Setting up your navigator, ahoy!");
    nav.init();

    loop {
        println!("{:?}", nav.fmt_debug());
        sleep(Duration::from_millis(1000));
    }
}
```

Fonte: elaborado pelo autor.

Figura 17 – Resultados da execução do código no terminal

```
blueos@raspberrypi-armv7:~/github/navigator-rs $ cargo run --example
raspberrypi
  Compiling navigator-rs v0.2.1 (/home/blueos/github/navigator-rs)
  Finished dev [unoptimized + debuginfo] target(s) in 7.79s
  Running `target/debug/examples/raspberry-pi`
Creating your navigator module!
Setting up your navigator, ahoy!
Navigator {
  adc: ADCData {
    channel: [
      0.036500003,
      0.0395,
      0.040000003,
      0.040000003,
    ],
  },
  bmp: Bmp {
    temperature: 34.05,
    altitude: 0.14004052,
    pressure: 101.049484,
  },
  imu: Imu {
    accelerometer: AxisData {
      x: 0.22027442,
      y: -0.241823,
      z: 9.7232,
    },
    gyroscope: AxisData {
      x: -0.004926848,
      y: 0.007057377,
      z: -0.0013315806,
    },
  },
  mag: AxisData {
    x: -15.6,
    y: 17.85,
    z: 12.450001,
  },
}
...
Continua o loop
...
```

Fonte: elaborado pelo autor.

### 7.3 BENCHMARK E TESTES

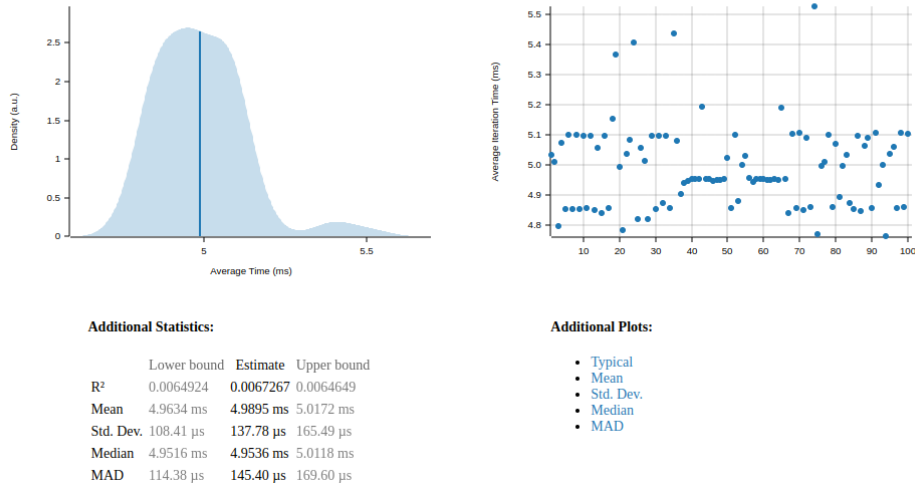
Ao finalizar a criação da biblioteca, iniciaram-se mais testes na Raspberry. Uma abordagem prática para tal finalidade envolveria a execução de um *benchmarking*, prática em que a função é repetidamente executada para aferir o tempo médio de execução do código. Esse procedimento é fundamental para validar o tempo de leitura dos sensores e realizar uma verificação abrangente do seu funcionamento.

Uma biblioteca que possibilita tais testes de *benchmark* é o projeto Criterion, que permite criar códigos para execução de testes focados em desempenho, que foi adotado neste trabalho e permitiu avaliar o tempo de execução das funções de leitura disponíveis na biblioteca.

Algumas análises serão brevemente apresentadas, como a do Gráfico 1, contendo o tempo de leitura do magnetômetro por meio da biblioteca ak09915-rs. O intervalo entre os limites inferior e superior sugere uma estimativa central relativamente precisa, com valor da média em torno de 4,99 ms, o período esperado para o magnetômetro configurado para leituras em 200 Hz, com um desvio padrão de aproximadamente 140  $\mu$ s.

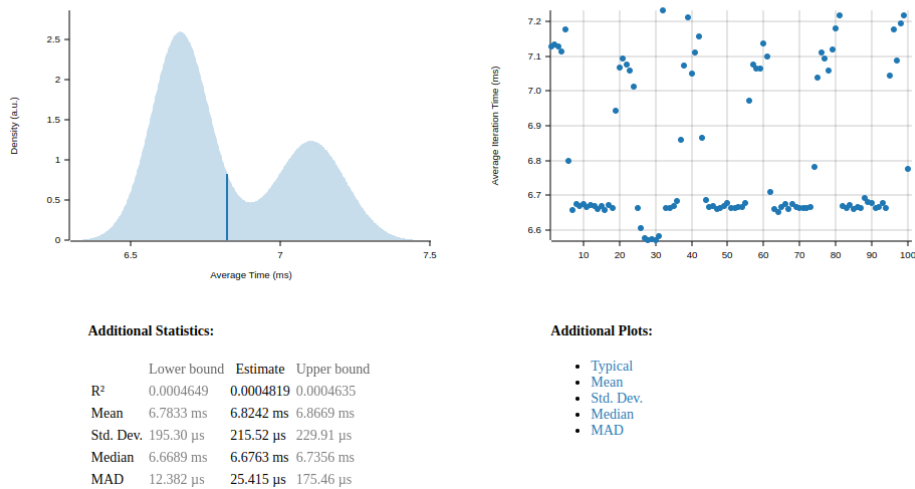
Já na função que realiza a medida em todos os sensores, “read\_all”, na análise disponível no Gráfico 2, pode-se observar um maior desvio, proveniente da acumulação de desvios em outros sensores. Tal desvio deverá ser apurado nas próximas versões da biblioteca.

Gráfico 1 – Gráficos gerados pelo Criterion – Leitura do magnetômetro read\_mag



Fonte: elaborado pelo autor.

Gráfico 2 – Gráficos gerados pelo Criterion – Leitura de todos sensores read\_all



Fonte: elaborado pelo autor.

## 7.4 CI/CD INTEGRAÇÃO

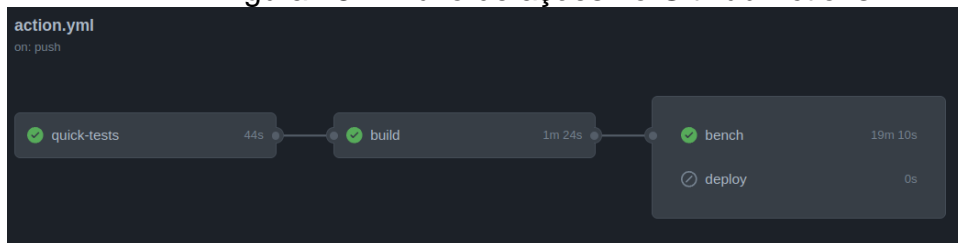
Como mencionado nos capítulos anteriores, um sistema de integração e entrega contínua foi criado para o gerenciamento deste projeto.

Na Figura 18, pode-se observar que a segmentação das tarefas executadas no GitHub Actions, tem quatro etapas:



1. “*Quick-test*” executa testes preliminares como formatação e checagem de boas práticas, ferramentas fornecidas pelo ambiente de desenvolvimento do Rust.
2. Em “*build*” há a compilação do código em modo *release* e a documentação é gerada.
3. Em “*bench*” é executado o código em Raspberry Pi 4 disponível, um servidor local é configurado, o *self-hosted runner*.
4. Após a conclusão e validação de todas as etapas anteriores, a etapa de entrega (também conhecida como “*deploy*”) é inicializada, gerando a biblioteca disponível no sistema do site crates.io.

Figura 18 – Fluxo de ações no GitHub Actions



Fonte: elaborado pelo autor.

## 7.5 PUBLICAÇÃO

A etapa de publicação irá disponibilizar a biblioteca no portal crates.io (Figura 19), no mesmo portal o usuário também consegue navegar até a documentação.

A documentação ficará atualizada a cada novo versionamento do código, um exemplo da página e demonstrado na Figura 20.

Figura 19 – Publicação do navigator-rs no crates.io

## navigator-rs v0.3.0 Follow

This crate serves as the entry point for embedding applications using Rust on Blue Robotics's Navigator

#embedded #robotics #bluerobotics #rov #navigator

---

Readme
6 Versions
Dependencies
Dependents
Settings

---

### Navigator-rs

This crate serves as the entry point for embedding applications using Rust on Blue Robotics's [Navigator](#).

The Navigator board has the Raspberry Pi HAT form factor, which allows you to easily attach it to a Raspberry Pi 4 board. Then you can unleash the power of Navigator to develop new solutions or interact with your ROV hardware.

The board offers the following capabilities:

Control:

- LEDs
- PWM (Pulse Width Modulation) with 16 channels

Measurements:

- ADC (Analog Digital Converter) entries
- Magnetic field
- Acceleration
- Angular velocity
- Temperature
- Pressure

#### Metadata

📅 17 days ago

📄 MIT

📦 19.8 KiB

#### Install

Run the following Cargo command in your project directory:

cargo add navigator-rs

Or add the following line to your Cargo.toml:

navigator-rs = "0.3.0"

#### Homepage

🔗 [bluerobotics.com/store/com...](#)


#### Documentation

🔗 [docs.bluerobotics.com/navig...](#)

#### Repository

Fonte: elaborado pelo autor.

Figura 20 – Documentação do navigator-rs



**Navigator**

**Methods**

- create
- fmt\_debug
- get\_led
- init
- new
- pwm\_enable
- read\_accel
- read\_adc
- read\_adc\_all
- read\_all
- read\_altitude
- read\_gyro
- read\_mag
- read\_pressure
- read\_temperature
- self\_test
- set\_led
- set\_led\_all
- set\_led\_toggle
- set\_neopixel
- set\_pwm\_channel\_value
- set\_pwm\_channels\_value
- set\_pwm\_channels\_val...
- set\_pwm\_freq\_hz
- set\_pwm\_freq\_prescale

```
pub fn self_test(&mut self) -> bool source
[-] pub fn pwm_enable(&mut self, state: bool) source
    Sets the PWM IC to be enabled through firmware and OE_pin.

    Arguments
    • state - The state of PWM output, it's enabled with a true logic value.

    Examples
    Please check set\_pwm\_channel\_value.

[-] pub fn set_pwm_channel_value(&mut self, channel: PwmChannel, value: u16) source
    Sets the Duty Cycle (high value time) of selected channel.
    On PCA9685, this function sets the OFF counter and uses ON value as 0.

    Further info
    Check 7.3.3 LED output and PWM control

    Examples
    1 use navigator_rs::{Navigator, PwmChannel};
    2
    3 let mut nav = Navigator::new();
    4
    5 nav.init();
    6 nav.pwm_enable(true);
    7
    8 nav.set_pwm_freq_prescale(99); // sets the pwm frequency to 60 Hz
    9 nav.set_pwm_channel_value(PwmChannel::Ch1, 2048); // sets the duty cycle to 50%

[-] pub fn set_pwm_channels_value<const N: usize>(
    &mut self,
    channels: &[PwmChannel; N],
    value: u16
)
    Like set\_pwm\_channel\_value. This function sets the Duty Cycle for a list of multiple channels.
```

Fonte: elaborado pelo autor.

## 8 DESENVOLVIMENTO – O PROJETO NAVIGATOR-LIB

Durante a execução do trabalho fora iniciado o projeto *multi-binding* cpy-rs, uma biblioteca utilizada para gerar bibliotecas para C++ e Python por meio de uma integração dos projetos *cbindgen* e *pyo3*. O projeto hoje está disponível na plataforma *crates.io* como seu código fonte também (GitHub, 2023).

Este projeto também contém em seu repositório um modelo de uso de fácil reprodução e rápida configuração para o usuário. O modelo entrega um sistema com compilação, testes e a entrega dos pacotes. A partir da execução das rotinas do modelo, obtêm-se os binários e *header* para a linguagem C e um pacote *wheel* para a linguagem Python.

Com o projeto de *multi-binding* realizado, foi iniciado, na BlueRobotics o projeto Navigator-lib, projeto que permitirá a disponibilização de bibliotecas para as linguagens Python e C++.

### 8.1 PYTHON

Python é uma linguagem popular, interpretada e acessível. Para realizar os primeiros testes é necessário criar um pacote de instalação *wheel*, que é facilmente obtido utilizando o modelo fornecido pelo projeto *cpy-rs*.

Depois de fazer a integração inicial do projeto Navigator-lib para usar o Navigator-rs e o *cpy-rs*, de maneira rápida e intuitiva foi gerado um pacote de instalação para a linguagem Python, o *Bluerobotics-navigator*.

Um dos desafios na confecção desta biblioteca foi a necessidade de utilizar um *singleton*, um padrão que permite a criação de um objeto apenas uma vez, algo necessário quando se tem um objeto como a Navigator, que precisa realizar configurações em sua inicialização.

Com o padrão *singleton* criado para um objeto Navigator, funções livres foram criadas e as definições feitas de forma que ficassem similares às já conhecidas na biblioteca em Rust.

As funções livres puderam ser executadas recebendo o Navigator guardado por um *mutex*, uma vez que a função é executada, ela ganha a utilização do Navigator. Esse processo continua até que o resultado seja obtido, permitindo que o Navigator

esteja disponível para reutilização, evitando possíveis execuções simultâneas, se ocorrerem.

### 8.1.1 Projeto

Para gerenciamento do pacote, dependências, ambientes de compilação e descrições um gerenciador de projeto se faz necessário. Uma das ferramentas mais comuns é o `pyproject`. Neste trabalho, em específico, escolheu-se o *hatch*, uma versão mais moderna adequada com as PEPs novas propostas pelo Python.

O *hatch* apresentou uma fácil usabilidade, pois foi possível definir ambientes como: compilação, desenvolvimento e documentação. O *hatch* cria instâncias com todos os pacotes necessários e definidos para estes ambientes.

### 8.1.2 Documentação

Para a documentação da biblioteca Python foram selecionadas duas ferramentas principais: o *sphinx* e o *napoleon*. Uma série de novas atualizações foi realizada na biblioteca `cpy-rs`, com as quais o projeto pôde gerar documentação para Python e C++ de forma separada.

## 8.2 C++

O lançamento de binários para C++ também foi possível utilizando o projeto `Cpy-rs`. Uma das dificuldades encontrada foi a incompatibilidade de algumas funções que possuíam equivalente em Python, mas não na linguagem C++.

Isso exigiu a introdução de novos parâmetros para definir comportamentos distintos para funções Python e para C++.

### 8.2.1 Compilação

Para compilar o projeto, disponibilizou-se de uma pasta de testes, que utiliza o `Cmake` para gerenciamento e compilação.

O projeto em Cmake realizado precisa de uma biblioteca dinâmica (arquivo com extensão “.so”) e de um cabeçalho identificando as funções(header), ambos providos pelo projeto Navigator-lib.

Com os arquivos gerados e dispostos na pasta de projeto do Cmake, resta o código a ser escrito pelo usuário utilizando a linguagem C++. Feita a compilação do projeto, o arquivo gerado poderá ser executado no Raspberry Pi 4.

### 8.2.2 Documentação

O arquivo de documentação para C++ foi provido junto com o *header*, além de ele também já conter as definições das funções.

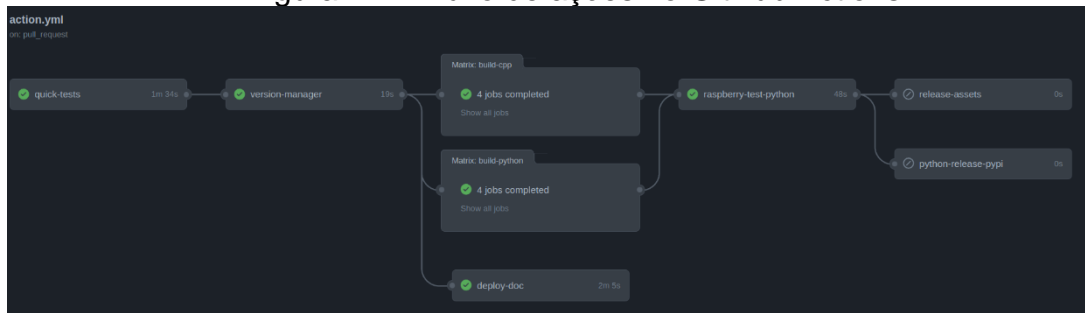
A geração do *header* também ficou integrada e totalmente automatizada nos processos. O *header* por sua vez é disponibilizado na página do projeto com os binários na categoria releases.

## 8.3 CI/CD INTEGRAÇÃO

Igualmente ao projeto Navigator-rs, uma sequência de etapas foi realizada para o acompanhamento do projeto, representado pela Figura 21 e detalhado a seguir:

1. “*Quick-test*” com mesmas finalidades mencionadas no projeto anterior.
2. “*Version manager*” que facilita o lançamento de versões futuras.
3. “*Build*” há a compilação do código em modo release para as duas linguagens e nas 4 arquiteturas mencionadas no trabalho, além de sua instalação em ambientes que emulam a arquitetura.
4. “*Deploy-doc*” confere e gera a documentação.
5. “*Raspberry-test*” executa o código em um Raspberry Pi 4 disponível, um servidor local é configurado, o *self-hosted runner*.
6. “*Deploy*” foi dividido em “*python-release*” e “*realize-assets*”.

Figura 21 – Fluxo de ações no GitHub Actions



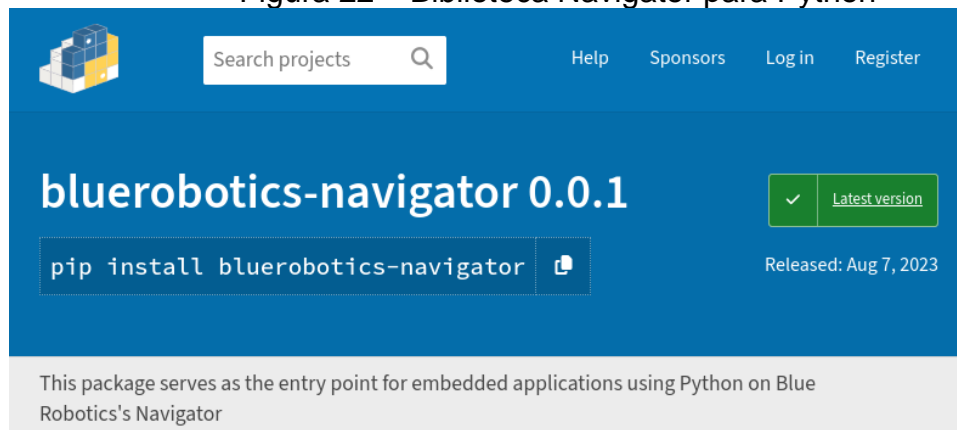
Fonte: elaborado pelo autor.

## 8.4 PUBLICAÇÃO

A publicação da biblioteca Python está disponível no PyPI (Figura 22), possibilitando a qualquer usuário a instalar pelo aplicativo Pip.

Na página do projeto, o usuário também consegue acessar a documentação (Figura 23) com exemplos e utilizá-la em seu ambiente.

Figura 22 – Biblioteca Navigator para Python



Fonte: PyPi (2023).

Figura 23 – Página com documentação para biblioteca Python

The screenshot shows the documentation page for the BlueRobotics's Navigator Library. The page has a blue header with the logo and the text 'bluerobotics-navigator' and '0.0.0'. Below the header is a search bar labeled 'Search docs'. The main content area is titled 'BlueRobotics's Navigator Library' and includes the following text:

This library serves as the entry point for embedding applications using Python on Blue Robotics's Navigator.

The Navigator board has the Raspberry Pi HAT form factor, which allows you to easily attach it to a Raspberry Pi 4 board. Then you can unleash the power of Navigator to develop new solutions or interact with your ROV hardware.

The board offers the following capabilities:

Control:

- LEDs
- PWM (Pulse Width Modulation) with 16 channels

Measurements:

- ADC (Analog Digital Converter) entries
- Magnetic field
- Acceleration
- Angular velocity
- Temperature
- Pressure

Currently, it supports `armv7` and `aarch64` architectures, which are the official defaults for BlueOS. However, despite using the embedded-hal concept, new ports can be added as long as the platform matches the hardware design and specifications.

For more information, including installation instructions, schematics, and hardware specifications, please check the navigator hardware setup [guide](#).

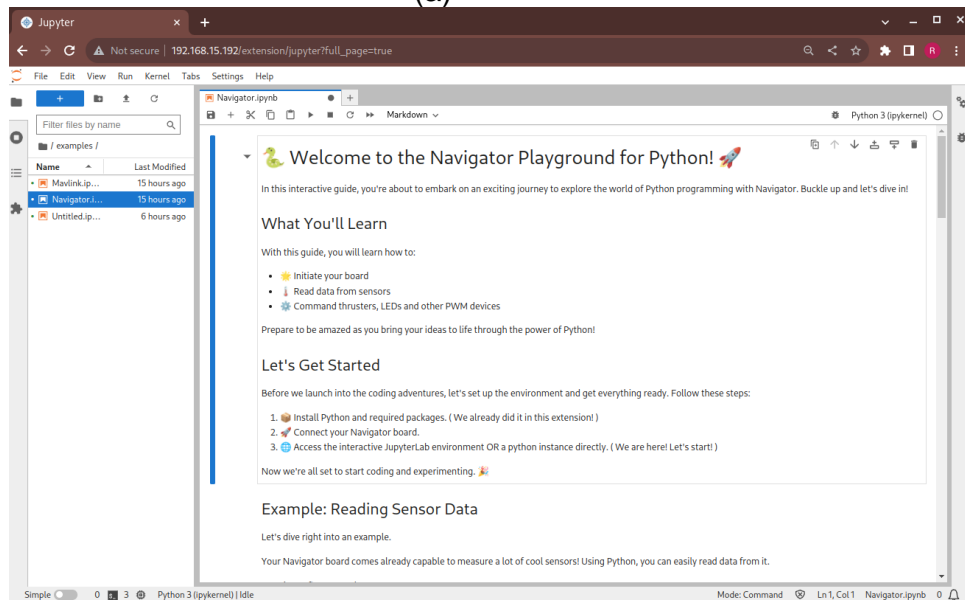
The code snippet shows the class `bluerobotics_navigator.ADCData` with a `channel` attribute.

Fonte: elaborado pelo autor.

## 8.5 EXTENSÃO DO BLUEOS

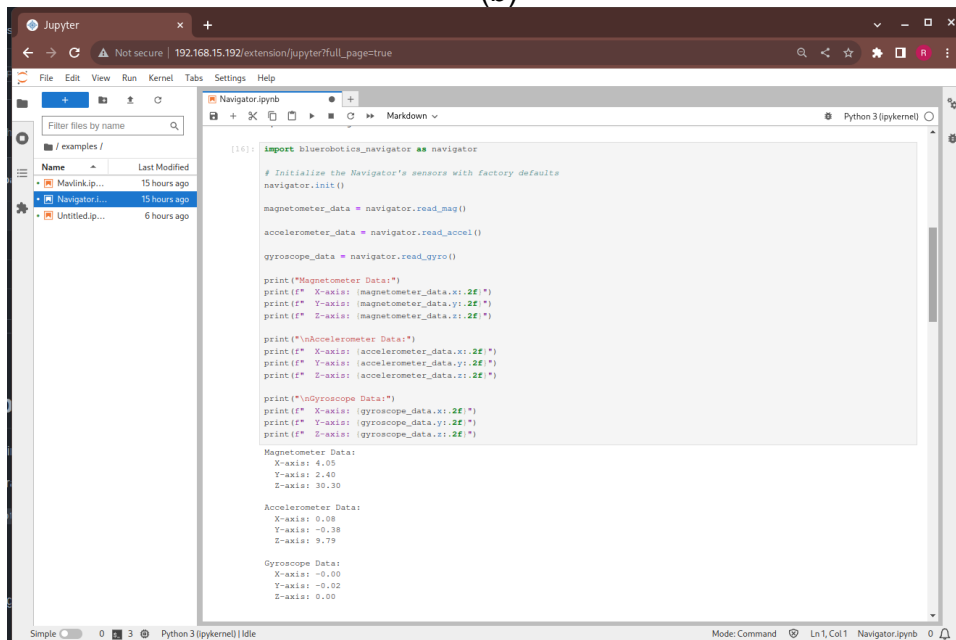
Uma extensão para o sistema BlueOS também foi criada. Trata-se de uma adaptação do projeto Jupyter-Lab, ambiente de programação Python acessível via web, com a biblioteca do Navigator previamente instalada. O resultado é demonstrado nas Figura 24, Figura 25, Figura 26.

Figura 24 – Exemplo de Extensão do BlueOS utilizando a biblioteca Python (a)



Fonte: elaborado pelo autor.

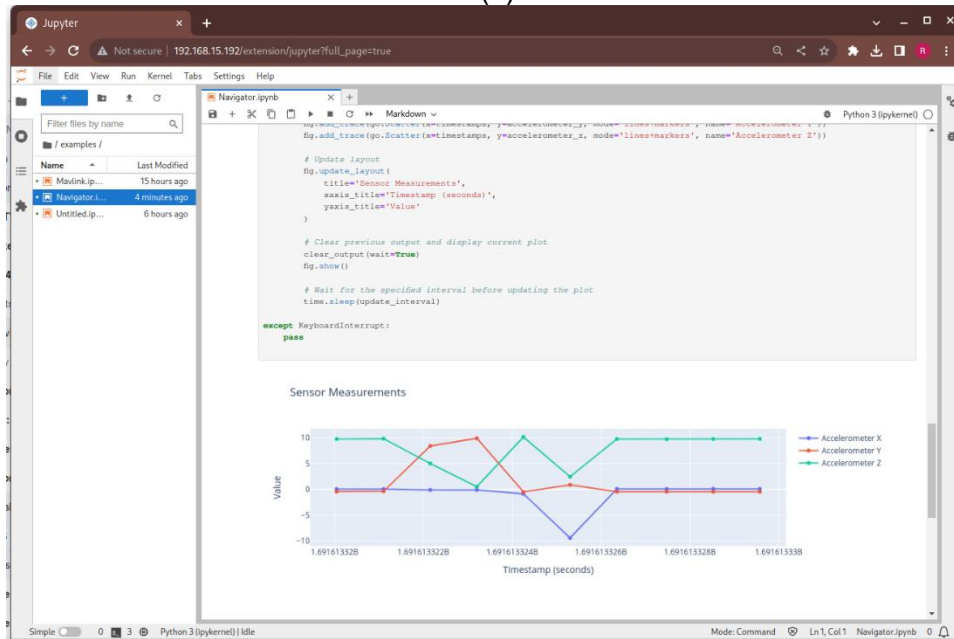
Figura 25 – Exemplo de Extensão do BlueOS utilizando a biblioteca Python (b)



Fonte: elaborado pelo autor.



Figura 26 – Exemplo de Extensão do BlueOS utilizando a biblioteca Python  
(c)



Fonte: elaborado pelo autor.

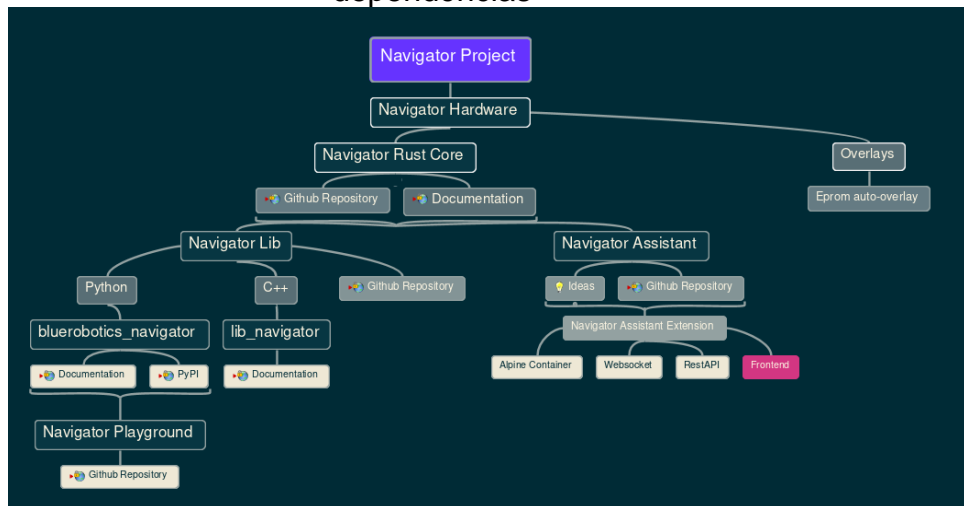
## 9 RESULTADOS OBTIDOS

Durante a realização deste trabalho, desenvolveram-se projetos que auxiliaram na realização final das bibliotecas. Destacaram-se aqui os projetos Navigator-rs e Navigator-lib, que realizam o trabalho proposto neste trabalho de conclusão de curso, além dos projetos cpy-rs e ak0915-rs também citados.

Este projeto foi estruturado de forma segmentada, pois utilizou-se o GitHub para versionamento e o GitHub Actions para os testes e automações. Dessa forma, caso sejam feitas melhorias na biblioteca de referência (Rust), o desenvolvedor consegue validar e testar rapidamente as mudanças. Essas melhorias podem ser estendidas a projetos subsequentes, como o Navigator-lib. O Navigator-lib, por sua vez, também passará por uma bateria de testes antes de se disponibilizarem as bibliotecas para Python e C++.

No diagrama dos projetos desenvolvidos (Figura 27), é possível analisar suas dependências e segmentação.

Figura 27 – Diagrama listando os projetos desenvolvidos e suas dependências

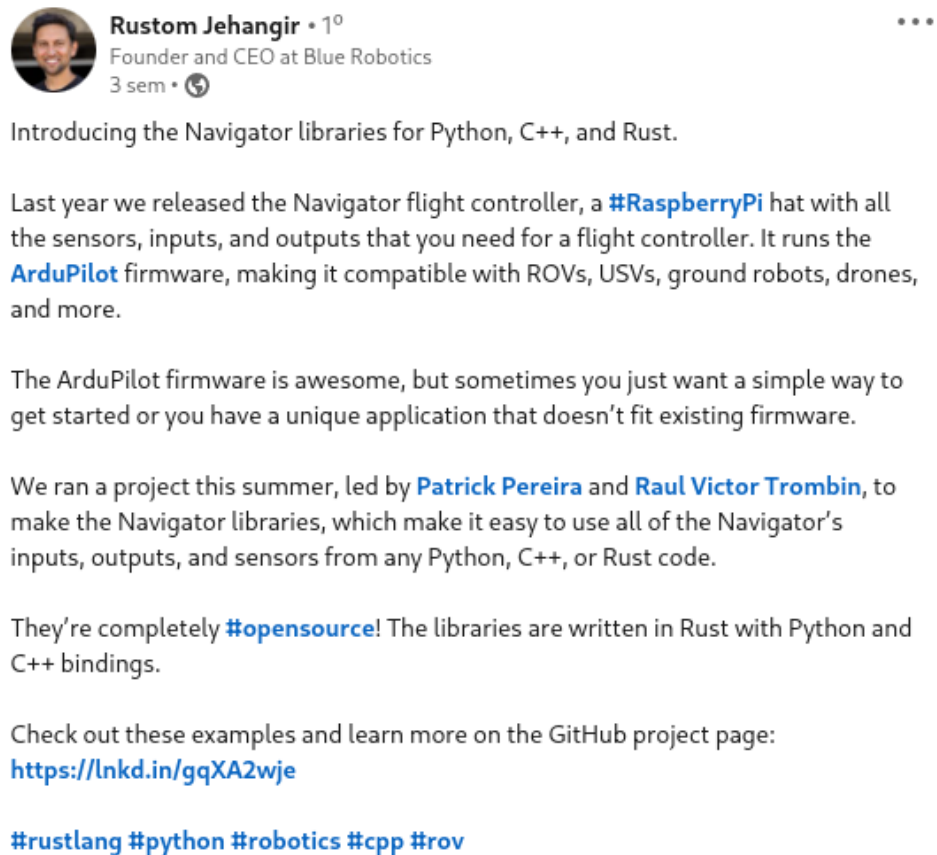


Fonte: elaborado pelo autor.

Após a realização da biblioteca Rust, Python e C++, estas foram anunciadas na rede social LinkedIn (

Figura 28).

Figura 28 – Anúncio das bibliotecas na rede social LinkedIn



**Rustom Jehangir** • 1º  
 Founder and CEO at Blue Robotics  
 3 sem • 🌐

Introducing the Navigator libraries for Python, C++, and Rust.

Last year we released the Navigator flight controller, a **#RaspberryPi** hat with all the sensors, inputs, and outputs that you need for a flight controller. It runs the **ArduPilot** firmware, making it compatible with ROVs, USVs, ground robots, drones, and more.

The ArduPilot firmware is awesome, but sometimes you just want a simple way to get started or you have a unique application that doesn't fit existing firmware.

We ran a project this summer, led by **Patrick Pereira** and **Raul Victor Trombin**, to make the Navigator libraries, which make it easy to use all of the Navigator's inputs, outputs, and sensors from any Python, C++, or Rust code.

They're completely **#opensource**! The libraries are written in Rust with Python and C++ bindings.

Check out these examples and learn more on the GitHub project page:  
<https://lnkd.in/gqXA2wje>

**#rustlang #python #robotics #cpp #rov**

Fonte: LinkedIn (2023).

Um anúncio mais específico foi realizado no fórum da BlueRobotics, como pode ser visto na Figura 29 e Figura 30. Na publicação, ressaltou-se a facilidade de programação do hardware nas três linguagens. Um trecho de programação em Rust (Figura 31), um trecho com Python (Figura 32) e C++ (Figura 33).

Figura 29 – Anúncio da biblioteca no fórum principal (a)

**Navigator Library!**

Blue Robotics Software BlueOS news, navigator, raspberry



Patrick José Pereira patrickelectric Blue Robotics Engineer

2 Aug 7

Hello Blue Robotics Community! 🙌

We're beyond excited to introduce you to the [Navigator Library](#) - a game-changer for all who wish to harness the full potential of the [Navigator](#) board!

Whether you're a **Python** aficionado or a **C++** guru, the Navigator Library is tailored for you. This library will empower you to light up LEDs, test temperatures, and sample the onboard sensors, all with simple functions!

For our **Rust** enthusiasts, don't feel left out! Dive into the [navigator-rs library](#) crafted just for you.

**🚀 How to use it**

Check the project [README](#) for details about how to install, use and create your own project from the beginning. But here are some minimal examples:

Fonte: BlueRobotics (2023).

Figura 30 – Anúncio da biblioteca no fórum principal (b)

**🗨️ Why a Navigator Library?**

The Navigator board has huge potential, and we want to unlock the hardware capabilities for a new range of users. Whether you're developing a new solution, interacting with your ROV hardware, or just experimenting, this library paves the way for endless possibilities. Now, you can create your own application without dealing with MAVLink messages or ArduSub firmware - your code can access the hardware directly!

We're looking forward to the waves you make with this new library! 🌊

Fonte: BlueRobotics (2023).

Figura 31 – Anúncio da biblioteca no fórum principal (c)

**🦀 Rust**

```
fn main() {
    let mut nav = Navigator::new();

    println!("Setting up your navigator, ahoy!");
    nav.init();

    nav.set_pwm_channel_value(PwmChannel::All, 0);
    println!("{:#?}", nav.fmt_debug());
}
```

Fonte: BlueRobotics (2023).

Figura 32 – Anúncio da biblioteca no fórum principal (d)

 Python

```
import bluerobotics_navigator as navigator

navigator.init()

# Turning on LED!
navigator.set_led(navigator.UserLed.Led1, True)

# Reading onboard temperature!
print(f"Temperature: {navigator.read_temp()}")

# Set connected RGB LED to blue
navigator.set_neopixel([[0, 0, 255]])
```

Fonte: BlueRobotics (2023).

Figura 33 – Anúncio da biblioteca no fórum principal (e)

 C++

```
int main() {
    printf("Initializing navigator module.\n");
    init();

    printf("Turning on LED!\n");
    set_led(UserLed::Led1, true);

    printf("Temperature: %f\n", read_temp());
    printf("Pressure: %f\n", read_pressure());

    ADCData adc = read_adc_all();
    printf("Reading ADC Channels: 1 = %f, 2 = %f, 3 = %f, 4 = %f\n",
        adc.channel[0], adc.channel[1], adc.channel[2], adc.channel[3]);
}
```

Fonte: BlueRobotics (2023).

## 10 CONCLUSÕES

Ao longo deste trabalho, foi possível obter um maior conhecimento da ciência envolvida nos ROVs, veículos estes que estão presentes em diversas áreas de pesquisa científica e em aplicações sofisticadas de engenharia.

Exploraram-se diversos aspectos nos diferentes domínios da Engenharia Elétrica. Analisaram-se diferentes habilidades requeridas para a execução de um projeto de engenharia envolvendo um dispositivo moderno e complexo, que atua em um ambiente inóspito, incluindo seu controle e operação.

Não somente os aspectos elétricos e eletrônicos de um mecanismo sofisticado precisam ser dominados, mas sua interface com sistemas de controle e sensores também necessita de conhecimentos técnicos específicos, unindo diversos campos de conhecimento.

Assim, uniram -se os conhecimentos de máquinas elétricas, circuitos elétricos e eletrônicos, sistemas de comunicação, *hardwares*, *softwares* e programação. Neste último tópico, destaca-se o aprendizado de técnicas e práticas profissionais de programação, teste e divulgação de *softwares* em plataformas internacionais.

Desenvolveu-se o conhecimento da linguagem Rust, que se encontra em tendência adotada por companhias de destaque na área de tecnologia. O conhecimento da linguagem e suas ferramentas viabilizou a realização dos trabalhos propostos pela empresa BlueRobotics, como a disponibilização de uma biblioteca para seu *hardware* em diferentes linguagens.

No decorrer deste processo, criou-se uma biblioteca principal escrita em Rust, chamada de Navigator-rs, na qual foram explorados os sensores e funcionalidades da plataforma. Neste projeto da biblioteca, obtiveram-se as garantias de segurança, desempenho e multi-compatibilidade exigidas para sistemas complexos de engenharia, o que valoriza sobremaneira o projeto.

Neste processo de desenvolvimento, também se realizou o lançamento de duas novas bibliotecas disponíveis em linguagens mais populares e comuns no mercado de robótica e ROV, o Python e o C++. Tais bibliotecas foram lançadas e disponíveis ao público sob o projeto Navigator-lib.

Todas essas bibliotecas lançadas dispõem de um versionamento de código, um controle de qualidade, testes e geração automática de documentação.

A publicação de novas versões também é automatizada, fazendo com que os projetos possam receber novas otimizações e realizações futuras.

Aprimoramentos possíveis seriam mais testes desempenho e a criação de uma nova biblioteca que realize a disponibilização dos sensores para aplicações web, projeto que se encontra em andamento na empresa.

Por fim, pretende-se que este Trabalho de Conclusão de Curso inspire outros estudantes a buscar a integração de conhecimentos multidisciplinares, unindo *hardware* e *software*, de forma a ganhar um diferencial profissional no mercado de trabalho.

## REFERÊNCIAS

ACTIVE Python Releases. **Python™**. 2023. Disponível em: <https://www.Python.org/downloads/> Acesso em: 26 nov. 2023.

ADAMS, J. **Introducing Raspiberry Pi HATs**. Raspberry Pi, 31 jul. 2014. Disponível em: <https://www.raspberrypi.com/news/introducing-raspberry-pi-hats/> Acesso em: 26 nov. 2023.

CHRIST, R. D.; WERNLI, R. L. **The Navy's CURV II vehicle**. [s.d]. p.1112. *In:\_\_\_\_\_*. **The ROV Manual: a user guide for observation-class remotely operated vehicles**. 1. ed. UK: Butterworth-Heinemann, jul. 2007. Disponível em: <https://ntcontest.ru/upload/iblock/52d/52d49cf6584942a6f5479d0402a35bf9.pdf>. Acesso em: 25 nov. 2023.

CHRIST, Robert D.; WERNLI, Robert L. **The ROV Manual: a user guide for observation-class remotely operated vehicles**. 2. ed. [S. L.]: Butterworth-Heinemann, 2018. 712 p.

CARDOSO, J. A.; PEREIRA, P. J.; TROMBIN, R. V. **Navigator-rs**. 2023a. Disponível em: <https://GitHub.com/bluerobotics/Navigator-rs>. Acesso em: 12 jun. 2023.

\_\_\_\_\_. **Navigator-lib**. 2023b. Disponível em: <https://GitHub.com/bluerobotics/Navigator-lib>. Acesso em: 12 jun. 2023.

\_\_\_\_\_. **Cpy-rs**. 2023c. Disponível em: <https://GitHub.com/patrickeletric/cpy-rs>. Acesso em: 12 jun. 2023.

CORRÊA, F. das G. Inovação em tecnologias submarinas na exploração oceânica: passado e presente. **Revista InterAção**, v. 12, n.2, 2022. pp. 13–27. <https://doi.org/10.5902/2357797553173>. Acesso em: 29 out. 2023.

DEVELOPER Survey. **Stack Overflow**. 2023. Disponível em: <https://survey.stackoverflow.co/2023/#overview>. Acesso em: 26 nov. 2023.

FERREIRA, A. W. L.; MARAVILHA, D. M. **Veículo Remotamente Operado (ROV): A Importância da Aplicação de Sua Tecnologia Para Produção Petrolífera**. 2008. Monografia (Tecnologia em Automação Industrial). Centro Federal de Educação Tecnológica de Campos (CEFET), Campos dos Goytacazes, RJ, 2011. Disponível em: <https://docplayer.com.br/30688568-Curso-de-tecnologia-em-automacao-industrial-arlen-wanderson-landim-ferreira-diego-machado-maravilha.html>. Acesso em: 29 out. 2023.

FISHERS, J. W. 4 tools that can make search and recovery operations successful. **Police1**, 18 fev. 2021. Disponíveis em: <https://www.police1.com/police-products/search-and-rescue/articles/4-tools-that-can-make-search-and-recovery-operations-successful-4B3aRXfDn7TNXaxo/> Acesso em: 27 nov. 2023.



FLEETMONITORING. **Fleetmonitoring**. 2023. Disponível em: <https://fleetmonitoring.euro-argo.eu/dashboard?Status=Active>. Acesso em: 25 nov. 2023.

GIRDHAR, Y. CUREE: A Curious Robot for Ecosystem Exploration. **ArpLab**, 7 mar. 2023. Disponível em: [https://warp.who.edu/curee/?\\_gl=1\\*xxbqh0\\*\\_ga\\*MTI4OTU5MzlwOC4xNjk2ODY3ODQ1\\*\\_ga\\_HLKfZX9JZk\\*MTY5NjkxNjQxMy43LjEuMTY5NjkxNzlyMi4wLjAuMA..\\*\\_gcl\\_au\\*NTc3NTg3NzYzLjE2OTY4Njc4NDA](https://warp.who.edu/curee/?_gl=1*xxbqh0*_ga*MTI4OTU5MzlwOC4xNjk2ODY3ODQ1*_ga_HLKfZX9JZk*MTY5NjkxNjQxMy43LjEuMTY5NjkxNzlyMi4wLjAuMA..*_gcl_au*NTc3NTg3NzYzLjE2OTY4Njc4NDA). Acesso em: 25 nov. 2023.

GitHub.PYO3/maturin. **GitHub**, 2023a. Disponível em: <https://github.com/PyO3/maturin> Acesso em: 4 dez. 2023.

GitHub.PYO3/pyo3. **GitHub**, 2023b. Disponível em: <https://github.com/PyO3/pyo3> Acesso em: 4 dez. 2023.

GITHUB DOCS. Entendendo o GitHub Actions. **GitHub Docs**. 2023a. Disponível em: <https://docs.github.com/pt/actions/learn-github-actions/understanding-github-actions>. Acesso em: 4 dez. 2023.

GITHUB DOCS. Sobre o Git. **GitHub Docs**. 2023b. Disponível em: <https://docs.GitHub.com/pt/get-started/using-git/about-git>. Acesso em: 26 nov. 2023.  
JAMSTEC. Video frame grab of the Japanese full ocean depth rated ROV Kaikō placing a flag to mark the deepest place on Earth; Challenger Deep, 10 911 m in the Mariana Trench. 24 mar.1995. 1 fotografia. *In*: JAMIESON, A. **The Hadal Zone: Life in the Deepest Oceans**. 2015. Disponível em: [https://assets.cambridge.org/97811070/16743/excerpt/9781107016743\\_excerpt.pdf](https://assets.cambridge.org/97811070/16743/excerpt/9781107016743_excerpt.pdf). Acesso em: 25 nov. 2023.

JAMIESON, A. **The Hadal Zone: Life in the Deepest Oceans**. 2015. Disponível em: [https://assets.cambridge.org/97811070/16743/excerpt/9781107016743\\_excerpt.pdf](https://assets.cambridge.org/97811070/16743/excerpt/9781107016743_excerpt.pdf). Acesso em: 25 nov. 2023.

LANGOWSKI, A.; LEBLANC, B. Announcing Windows 11 Insider Preview Build 25905. **Windows Blog**, 12 jul. 2023. Disponível em: <https://blogs.windows.com/windows-insider/2023/07/12/announcing-windows-11-insider-preview-build-25905/#:~:text=Rust%20in%20the%20Windows%20Kernel> Acesso em: 26 nov. 2023.

MATTHEWS, B. **Code like a Pro in Rust**. New York: Manning Publications, 2022.

MAYER, L. *et al.* The Nippon Foundation—GEBCO Seabed 2030 Project: The Quest to See the World's Oceans Completely Mapped by 2030. **Geosciences**, v.8, n.63, fev. 2018. Online. Disponível em: <http://dx.doi.org/10.3390/geosciences8020063>. Acesso em: 24 nov. 2023.

Mozilla. **Cbindgen**. 2023. Disponível em: <https://GitHub.com/mozilla/cbindgen>. Acesso em: 27 jun. 2023.

NAKANISH, H.; HASHIMOTO, J. (2011). A precise bathymetric map of the world's deepest seafloor, Challenger Deep in the Mariana Trench. *Marine Geophysics Research*, 2011. pp. 455-463,

PLATFORM Support. **The rustc book**. 2023. Disponível em: <https://doc.rust-lang.org/nightly/rustc/platform-support.html>. Acesso em: 26 nov. 2023. "The Rust Programming Language". Rust Foundation, <https://doc.rust-lang.org/book/>. Acesso em: 15 jun. 2023.

PyPi. Bluerobotics-navigator 0.0.2. **PyPi**. 2023. Disponível em: <https://pypi.org/project/bluerobotics-navigator/> Acesso em: 26 nov. 2023.

SEABED 2030. Our mission. 2023. Disponível em: <https://seabed2030.org/our-mission/>. Acesso em: 14 nov. de 2023. BLUEROV2. **Blue Robotics**. [s.l.], 2023. 1 fotografia. Disponível em: <https://bluerobotics.com/store/rov/bluerov2/>. Acesso em: 25 nov. 2023.

SMITH, W.H.; SANDWELL, D.T. Global Sea floor topography from satellite altimetry and ship depth soundings. **Science**, v. 277, n. 5334, 1997. pp.1956-1962.

STOEP, J. V. Memory Safe Languages in Android 13. **Google Security Blog**, 1 dez. 2022. Disponível em: <https://security.googleblog.com/2022/12/memory-safe-languages-in-android-13.html>. Acesso em: 12 ago. 2023.

VON LÖWIS, Martin. PEP 384 – Defining a Stable ABI. Status: Final. **Python Enhancement Proposals**. Disponível em: <https://peps.python.org/pep-0384/>. Acesso em: 26 nov. 2023.

WHAT is CI/CD? Continuous Integration & Continuous Delivery. **Katalon**. 2023. Disponível em: <https://katalon.com/resources-center/blog/ci-cd-introduction> Acesso em: 26 nov. 2023.

WALBRAN, A. Bare Metal Rust in Android. **Google Security Blog**, 9 out. 2023. Disponível em: <https://security.googleblog.com/2023/10/bare-metal-rust-in-android.html>. Acesso em: 15 out. 2023. Acesso em: 29 nov. 2023.

ROV Jason/Medea. **Woods Hole Oceanographic Institution**, [s.d.] 2023. Disponível em: Acesso em: <https://www.whoi.edu/what-we-do/explore/underwater-vehicles/ndsf-jason/> 4 dez. 2023.

Woods Hole Oceanographic Institution (WHOI). Magma Explosions feeding lava flows on West Mata Volcano. 2009. 1 vídeo (39s). *In*: PIECUCH, H. 3 memorable Jason Dives, **Ocean Tech**, 28 set. 2023. Disponível em: <https://www.whoi.edu/oceanus/feature/3-memorable-jason-dives/> Acesso em: 25 nov. 2023.