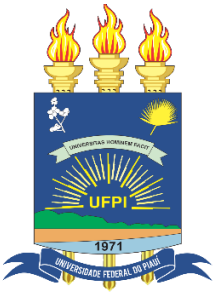

UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI

CENTRO DE CIÊNCIAS DA NATUREZA - CCN

DEPARTAMENTO DE COMPUTAÇÃO

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FRANCISCO SARMENTO NETO

Relatório: RoverLib

Teresina

Dezembro – 2025

Sumário

1. INTRODUÇÃO

O desenvolvimento de sistemas robóticos móveis exige a integração eficiente entre módulos de hardware e software, incluindo controle de motores, aquisição de imagens e processamento visual. Neste contexto, foi desenvolvida a biblioteca *RoverLib 0.1.0*, uma biblioteca em Python voltada ao controle e à manipulação de um Rover 1.0 equipado com visão computacional, executando sobre uma Raspberry Pi 5.

A biblioteca tem como objetivo abstrair o acesso ao hardware e fornecer uma interface modular para controle de movimento, aquisição de imagens, processamento visual e navegação autônoma do robô. A biblioteca RoverLib foi disponibilizada publicamente em um repositório GitHub, permitindo acesso ao código-fonte, versionamento e colaboração no desenvolvimento do projeto. O repositório pode ser acessado em: Clique aqui.

2. ARQUITETURA GERAL DA BIBLIOTECA

O projeto está contido no diretório raiz **lib_rover**, que abriga tanto os arquivos de empacotamento quanto o código-fonte principal da biblioteca.

```
lib_rover/
├── rover_lib/
│   ├── configs/
│   ├── modules/
│   │   ├── camera/
│   │   ├── movement/
│   │   ├── processing/
│   │   └── vision/
│   ├── utils/
│   ├── rover.py
│   └── __init__.py
├── README.md
└── setup.py
```

2.1. Organização dos diretórios

DIRETÓRIO / ARQUIVO	FUNÇÃO
setup.py	Empacotamento e distribuição da biblioteca

configs/	Armazenamento de parâmetros do sistema
modules/camera	Aquisição de imagens
modules/processing	Pré-processamento de imagens
modules/vision	Algoritmos de visão computacional
modules/movement	Controle dos motores
utils/	Utilidades gerais (configuração, suporte)
rover.py	Classe principal de integração

A RoverLib foi estruturada de forma modular, permitindo independência entre os subsistemas de câmera, visão computacional e movimento. Essa abordagem facilita manutenção, testes e futuras extensões da biblioteca.

3. EMPACOTAMENTO DA BIBLIOTECA (setup.py)

O arquivo **setup.py** é responsável pelo empacotamento e distribuição da biblioteca, possibilitando sua instalação por meio do **pip**, gerenciador de pacotes padrão da linguagem Python. Esse mecanismo permite a organização, versionamento e instalação automatizada da biblioteca em diferentes ambientes de desenvolvimento.

A biblioteca foi desenvolvida para operar com **Python versão 3.9 ou superior** e possui como dependências principais as bibliotecas **NumPy** e **OpenCV**, amplamente empregadas em aplicações de **visão computacional** e **robótica móvel**, especialmente em sistemas embarcados.

- Dados da Biblioteca:

- **Nome:** rover-lib

- **Versão:** 0.1.0

- **Dependências:** numpy; opencv-python

- **Linguagem:** Python 3.9 ou superior

- **Sistema operacional:** Linux (POSIX)

- **Plataforma-alvo:** Raspberry Pi (ambiente Linux embarcado)

- Observações:

Bibliotecas específicas da Raspberry Pi, como picamera2 e RPi.GPIO, **não estão listadas automaticamente**, pois devem ser instaladas diretamente no ambiente da Raspberry Pi.

4. SISTEMA DE CONFIGURAÇÃO (config.yaml)

O sistema de configuração da biblioteca **rover-lib** foi projetado com o objetivo de separar parâmetros de hardware e calibração do código-fonte, promovendo maior flexibilidade, manutenibilidade e reutilização do software. Para isso, foi adotado o formato **YAML** (YAML Ain't Markup Language), amplamente utilizado em sistemas embarcados e aplicações de robótica devido à sua legibilidade e fácil edição.

A utilização de um arquivo de configuração externo evita que valores críticos de hardware fiquem *hardcoded* no código, permitindo ajustes diretos conforme o ambiente, o dispositivo ou o cenário experimental, sem a necessidade de recompilar ou modificar a implementação da biblioteca.

4.1. Arquivo de configuração em yaml

Os parâmetros relacionados à câmera, aos pinos GPIO e à calibração dos motores são definidos em um arquivo YAML localizado no diretório **configs/config.yaml** da biblioteca. Esse arquivo é carregado dinamicamente pelo módulo **config**, garantindo que todas as partes do sistema utilizem uma configuração centralizada e consistente. Os grupos de parâmetros configuráveis são:

A) Parâmetros da câmera:

O arquivo de configuração permite definir características operacionais da câmera utilizada pelo Rover, como:

- **Resolução de captura** (ex.: 3280×2464), utilizada para aquisição de imagens em alta qualidade;
- **Resolução de pré-visualização** (ex.: 640×480), otimizada para processamento em tempo real;
- **Taxa de quadros por segundo (FPS)**, ajustável conforme a capacidade do hardware e a necessidade da aplicação.

Esses parâmetros possibilitam equilibrar qualidade de imagem e desempenho computacional.

B) Configuração dos pinos:

Os pinos responsáveis pelo controle dos motores são definidos no arquivo YAML, permitindo fácil adaptação a diferentes montagens físicas do Rover ou alterações no circuito eletrônico.
Por exemplo, são especificados:

- Pinos de controle do **motor esquerdo**;
- Pinos de controle do **motor direito**.

Essa abordagem torna o sistema independente de uma pinagem fixa, facilitando manutenções e substituições de componentes.

C) Parâmetros de calibração dos motores:

O sistema também contempla parâmetros de calibração, como:

- Limiar mínimo de acionamento dos motores;
- Ajustes finos para compensar diferenças mecânicas ou elétricas entre os motores.

Esses valores são fundamentais para garantir movimentação uniforme do Rover e podem ser ajustados experimentalmente, sem intervenção direta no código.

5. MÓDULOS DESENVOLVIDOS NA ROVER-LIB

A biblioteca rover-lib foi projetada de forma modular, separando claramente as responsabilidades de aquisição de imagens, processamento, visão computacional e controle de movimento. Essa organização facilita a manutenção, testes isolados e a evolução do sistema.

5.1. Módulo de Câmera (modules/camera)

O módulo camera é responsável pela **aquisição de imagens**, fornecendo frames de forma padronizada para os módulos de processamento e visão computacional.

5.1.1. cameraModule.py

Este módulo implementa a classe cameraModule, responsável por:

- Inicializar e configurar a câmera da Raspberry Pi utilizando a biblioteca picamera2;
- Definir resolução, formato de imagem e parâmetros de exposição;
- Capturar frames em tempo real;
- Converter automaticamente o formato de cores de RGB para BGR, garantindo compatibilidade com o OpenCV;
- Liberar corretamente os recursos de hardware ao final da execução.

Além disso, o módulo prevê tratamento de exceções e suporte a ambientes de desenvolvimento sem acesso direto à câmera física.

5.1.2. webcam.py

O arquivo webcam.py implementa uma alternativa de captura de imagens utilizando webcams convencionais via OpenCV (VideoCapture).
Essa classe é destinada principalmente a:

- Desenvolvimento e testes em ambientes não embarcados;
- Simulação do comportamento da câmera do Rover;
- Depuração de algoritmos sem necessidade do hardware final.

5.2. Módulo Movement (modules/movement)

O módulo movement concentra toda a lógica relacionada ao controle dos motores e movimentação do Rover, abstraindo os detalhes de baixo nível do hardware.

5.2.1. motor.py

Responsável pela interface direta com os motores, incluindo:

- Controle de velocidade por meio de PWM;
- Acionamento dos motores para frente, ré e parada;
- Encapsulamento da lógica de controle individual de cada motor.

5.2.2. motorCalibration.py

Este módulo implementa rotinas de calibração dos motores, permitindo:

- Ajustar limiares mínimos de acionamento;
- Compensar diferenças de torque ou resposta entre motores;
- Garantir movimentos mais simétricos e previsíveis.

Os parâmetros utilizados são obtidos a partir do sistema de configuração em YAML.

5.2.3. robot.py

O arquivo robot.py implementa a classe Robot, que atua como um controlador de alto nível da movimentação, sendo responsável por:

- Integrar os motores esquerdo e direito;
- Executar comandos como mover, parar e ajustar velocidades;
- Servir como base para algoritmos de navegação, como seguir linha ou desviar de obstáculos.

5.2.4. Subdiretório exceptions

Este diretório é reservado para exceções personalizadas relacionadas ao sistema de movimentação, permitindo tratamento de erros mais claro e específico, como falhas de inicialização ou controle de motores.

5.3. Módulo Processing (modules/processing)

O módulo processing é responsável pelo pré-processamento das imagens, preparando os frames para análise pelos algoritmos de visão computacional.

5.3.1. processing_image.py

Este módulo implementa métodos para:

- Ajuste de iluminação por correção gama;
- Segmentação de cores no espaço HSV;
- Operações morfológicas para remoção de ruído;
- Detecção de bordas utilizando o filtro de Canny.

Essas operações são fundamentais para melhorar a qualidade das informações extraídas da imagem, especialmente em ambientes com variações de iluminação.

5.4. Módulo Vision (modules/vision)

O módulo vision concentra os algoritmos de visão computacional, responsáveis por interpretar os frames capturados e gerar informações úteis para a navegação do Rover.

5.4.1. visionModule.py

Este módulo implementa a classe visionModule, responsável por:

- Detecção de círculos utilizando a Transformada de Hough;
- Detecção alternativa de círculos baseada em contornos e circularidade;
- Algoritmos de seguimento de linha, com cálculo do desvio em relação ao centro;
- Detecção de obstáculos baseada em cor e área mínima;
- Retorno de informações estruturadas para os módulos de controle de movimento.

5.5. Arquivos _init_.py

Os arquivos __init__.py presentes em cada diretório indicam que esses diretórios devem ser tratados como pacotes Python, permitindo importações organizadas e facilitando o empacotamento da biblioteca.

6. CLASSE PRINCIPAL (rover.py)

A classe Rover constitui o núcleo de coordenação da biblioteca rover-lib, sendo responsável por integrar e orquestrar os módulos de movimentação, aquisição de imagens e visão computacional. Seu objetivo é fornecer uma interface de alto nível para o controle do robô móvel, abstraindo os detalhes de hardware e dos algoritmos internos.

6.1. Estrutura geral da classe

A classe importa e integra os seguintes módulos fundamentais:

- Robot: controle de motores e movimentação;
- CameraModule: captura de imagens via câmera;
- VisionModule: processamento de imagens e tomada de decisão baseada em visão computacional;
- Config: leitura centralizada dos parâmetros de configuração a partir de arquivos YAML.

Essa organização permite que a classe Rover atue como um controlador central, responsável pela lógica de execução do sistema.

7. DIFICULDADES ENFRENTADAS

Ao realizar capturas de imagens de uma bola vermelha, observou-se que, em condições de iluminação ambiente intensa, ocorria um fenômeno de reflexão especular. Nesse caso, o ponto de maior incidência luminosa sobre a superfície da bola apresentava-se com coloração branca ou muito clara, devido à saturação do sensor da câmera.

Como consequência direta, ao aplicar a segmentação por cor no espaço HSV, a região altamente iluminada não era classificada como pertencente à faixa de cor vermelha. Isso resultava em falhas na máscara binária, caracterizadas por:

- “Buracos” no interior da região segmentada;
- Fragmentação do objeto detectado;
- Perda de continuidade da circunferência;
- Redução da confiabilidade do algoritmo de detecção de círculos.

Esse efeito comprometia significativamente o desempenho da Transformada de Hough para círculos, uma vez que o método depende da presença de bordas contínuas para o correto acúmulo de votos no espaço de parâmetros.

7.1. Estratégia de correção adotada

Para mitigar os efeitos da reflexão luminosa e restaurar a integridade da região segmentada, foi implementado um pipeline de pré-processamento morfológico e filtragem espacial, aplicado diretamente sobre a máscara binária gerada no espaço HSV.

As principais etapas adotadas foram:

7.1.1. Operações Morfológicas

- Abertura morfológica (MORPH_OPEN): utilizada para remover pequenos ruídos isolados;
- Fechamento morfológico (MORPH_CLOSE): aplicada para preencher falhas internas e unir regiões fragmentadas da máscara.

Essas operações contribuíram para recompor a forma do objeto segmentado, preenchendo as regiões afetadas pela reflexão da luz.

7.1.2. Suavização com Filtro Gaussiano

- Aplicação de um desfoque Gaussiano (Gaussian Blur) para suavizar transições abruptas na máscara;
- Redução de irregularidades nas bordas antes da aplicação da Transformada de Hough.

Essa etapa foi fundamental para fornecer uma imagem mais contínua e adequada ao processo de votação do acumulador.

8. PRÓXIMOS PASSOS

- **Aprimorar a documentação da biblioteca:** A biblioteca desenvolvida foi armazenada em um repositório na plataforma GitHub. Como melhoria futura, propõe-se a adição de arquivos README.md em cada módulo da biblioteca, com o objetivo de documentar de forma clara e detalhada a funcionalidade, a estrutura e o uso de cada componente. Essa iniciativa visa tornar o projeto mais compreensível, acessível e manutenível, facilitando tanto a utilização por novos usuários quanto a continuidade do desenvolvimento por outros colaboradores.

- **Desenvolver novos módulos de software:** Como próximo passo, propõe-se a integração de sensores ultrassônicos e LIDAR ao Rover, acompanhada do desenvolvimento de módulos de software compatíveis. Essa ampliação permitirá uma navegação mais estável, com movimentos mais suaves, curvas menos abruptas e melhor controle de posicionamento. Além disso, possibilitará o ajuste dinâmico da visão do robô, mantendo o objeto de interesse centralizado no campo de visão da câmera.

9. CONCLUSÃO

Ao longo do período de atividades realizadas até o mês de dezembro, foi possível o desenvolvimento da primeira versão da biblioteca “RoverLib**”**, concebida como o núcleo de software responsável por abstrair, organizar e integrar os diferentes subsistemas do Rover 1.0. A biblioteca viabiliza a comunicação entre hardware e software por meio de uma arquitetura modular, extensível e de fácil manutenção.

A RoverLib permitiu encapsular funcionalidades complexas, como controle de motores, aquisição de imagens, processamento e interpretação visual, em módulos bem definidos e independentes. Essa abordagem não apenas simplificou o uso do sistema, como também estabeleceu uma base sólida para futuras expansões, reutilização de código e integração de novos sensores e algoritmos. A implementação da primeira aplicação de visão computacional diretamente sobre a biblioteca demonstrou sua eficiência como camada de abstração e seu potencial como plataforma de desenvolvimento robótico.

10. BIBLIOGRAFIA

- **BRADSKI, Gary; KAEHLER, Adrian.**
  Learning OpenCV: Computer Vision with the OpenCV Library. Sebastopol: O’Reilly Media, 2008.

- **BRADSKI, Gary.**
  The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*, 2000.

- **GONZALEZ, Rafael C.; WOODS, Richard E.**
  Digital Image Processing. 4. ed. New York: Pearson, 2018.

- **HOUGH, P. V. C.**
  Method and means for recognizing complex patterns. *U.S. Patent 3,069,654*, 1962.

- **DALAL, Navneet; TRIGGS, Bill.**
  Histograms of Oriented Gradients for Human Detection. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.