🛠️ Manual do Projeto Prático de Software - Trainee EDRA 2025.1

Bem-vindo(a) ao Projeto Prático de Controle e Sistemas Embarcados!

Este manual detalha as opções de projeto para a área de Controle e Sistemas Embarcados do processo trainee da EDRA. Escolha **uma** das opções abaixo e siga as instruções para desenvolver seu projeto.

Opções de Projeto Detalhadas

1️⃣ Opção 1: Visão Computacional para Detecção de Formas Geométricas Coloridas

Objetivo: Desenvolver um programa em Python com OpenCV para detectar features geométricos de uma base de pouso de drones com cores predefinidas em imagens ou vídeos. O projeto visa propor o estudo de conceitos fundamentais da visão computacional como o uso de filtros de imagem, operações no espaços de cores, transformações morfológicas, thresholding, detecção de bordas e criação e tipificação de contornos utilizando OpenCV.

📚 **Materiais de Estudo:**

- **OpenCV Documentação e Tutoriais:**
  - Documentação Oficial OpenCV
    - **Uso:** Referência completa da biblioteca OpenCV, útil para pesquisa detalhada de funções e módulos.
  - Tutoriais OpenCV-Python: Focar em: Basic Image Operations, Image Filtering, Morphological Transformations, Color Spaces, Image Thresholding, Contours e GUI Features.
    - **Uso:** Guia prático com exemplos em Python, ideal para aprender os fundamentos do processamento de imagens com OpenCV. Concentre-se nos seguintes módulos:
      - **Basic Image Operations:** Operações básicas como leitura, escrita e manipulação

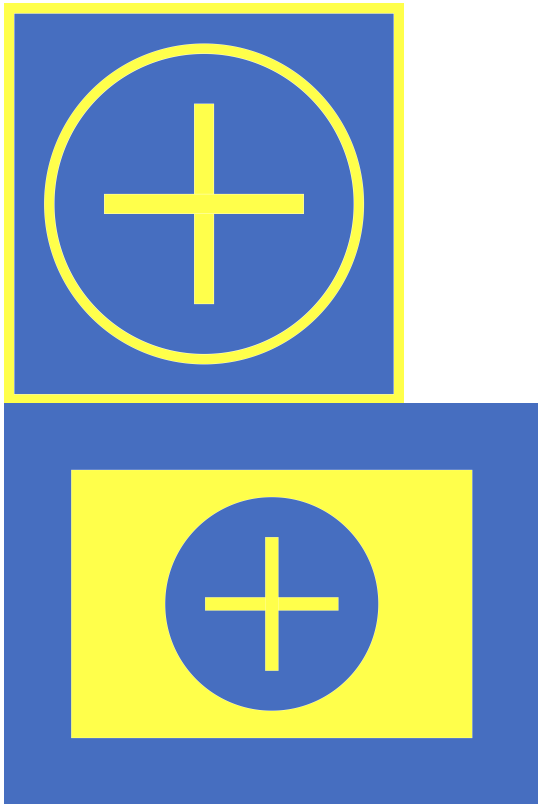
de pixels em imagens. Essencial para começar a trabalhar com imagens no OpenCV.
        - **Image Filtering:** Técnicas para suavizar imagens, remover ruído e realçar bordas. Inclui filtros como Gaussian Blur e Median Blur, importantes para pré-processamento.
        - **Morphological Transformations:** Operações morfológicas como erosão, dilatação, abertura e fechamento. Úteis para refinar formas geométricas detectadas e remover pequenos ruídos ou imperfeições.
        - **Color Spaces:** Conversão entre diferentes espaços de cor (RGB, HSV, Gray). O espaço HSV é particularmente útil para segmentação de cores, pois separa a informação de cor (Hue) da intensidade (Value) e saturação (Saturation).
        - **Image Thresholding:** Técnicas para segmentar imagens baseadas em limiares, criando imagens binárias. Essencial para isolar regiões de interesse baseadas em cor ou intensidade.
        - **Contours:** Detecção e análise de contornos em imagens binárias. Funções como `findContours` e `approxPolyDP` são cruciais para identificar e aproximar formas geométricas.
        - **GUI Features:** Ferramentas de Interface Gráfica do Usuário, como sliders e trackbars. Permitem criar interfaces interativas para ajustar parâmetros em tempo real.
    - Tutoriais de Processamento de Imagem OpenCV-Python
        - **Uso:** Tabela de conteúdos detalhada dos tutoriais de processamento de imagem em Python, facilitando a navegação por tópicos específicos.
    - Exemplo de interface com parâmetros para teste
        - **Uso:** Referência visual de uma interface de usuário interativa para ajuste de parâmetros em um detector de formas geométricas. Demonstra uma sequência de operações de visão computacional e como os parâmetros podem ser controlados via interface.
    - Exemplo de detecção de formas geométricas com OpenCV
        - **Uso:** Tutorial prático de detecção de formas geométricas básicas (triângulos, quadrados, círculos) em imagens usando OpenCV e Python. Útil para entender a detecção de contornos e formas.
- **Conceitos Fundamentais:**
    - **Filtros de Imagem:** Compreender filtros como Gaussian Blur, Median Blur para redução de ruído e suavização de imagens, facilitando a detecção de formas ao remover detalhes desnecessários e imperfeições.
    - **Morfologia Matemática:** Estudar operações de erosão, dilatação, abertura e fechamento

para refinar formas detectadas, remover ruídos menores, preencher lacunas em contornos e separar objetos que se tocam.

- **Segmentação por Cor:** Aprender a converter espaços de cor (RGB, HSV) e usar *thresholding* para isolar cores específicas, permitindo focar em regiões da imagem que correspondam às cores da base de pouso. O HSV é especialmente útil por separar a cor da luminosidade, tornando a detecção mais robusta a variações de iluminação.
- **Detecção de Contornos e Formas:** Utilizar `findContours` e `approxPolyDP` para detectar e aproximar contornos de formas geométricas, permitindo identificar as formas (círculos, quadrados, cruzes) presentes na base de pouso a partir dos contornos extraídos da imagem segmentada.

🎯 **Requisitos Funcionais:**

- **Requisitos Obrigatórios:**
  i. **Detecção de Formas Geométricas:** O programa deve detectar as formas geométricas básicas (círculos, quadrados, cruzes) contidas na base de pouso determinada (foto da definição abaixo no corpo do texto, uma pasta com mais fotos de angulos diversos para exemplo será disponibilizada) em imagens e vídeos, o programa deve ser capaz de associar um contorno e um ponto central para cada detecção. A detecção pode ser das formas básicas individualmente ou do conjunto definido pela união delas.



  ii. **Parametrização dos argumentos usados:** As formas geométricas a serem detectadas devem ter cores específicas (e.g., círculos e cruzes amarelos, quadrados azuis ou

quadrados amarelos etc), o programa deve lidar com isso internamente com parâmetros, assim como deve também parametrizar fatores fundamentais de filtros e qualquer argumento do tipo que for usada no código (permitir mudar em um unico lugar de maneira simples os valores usados, mesmo que atribua um valor padrão), tanto para a conveniência dos seus testes, para descobrir a configuração traz os melhores resultados, quanto para a demonstração que será feita.

iii. **Interface de Usuário (UI) Básica:** Utilizar elementos de GUI (Interface Gráfica do Usuário) do OpenCV como sliders, select e afins para ajustar em tempo real parâmetros de filtros e algoritmos (limites HSV, tamanho do kernel, blur etc) usados, que foram criados de maneira conveniente para isso.

iv. **Visualização:** Exibir a imagem original com as formas geométricas detectadas e contornadas, mostrando o tipo de forma e a cor detectada (texto sobreposto na forma ou na janela).

- **Requisitos Opcionais (Escolher pelo menos 2):**
  i. **Detecção em algum dataset próprio:** Criar um conjunto de fotos e vídeos (pode-se usar simplesmente a webcam ao vivo detectando numa tela ou numa impressão no papel) com uma réplica da base (pode ser em tamanho reduzido) esteja visível (é interessante criar diferentes condições para os testes) e validar o seu algoritmo para esse exemplo também.

  ii. **Reconhecimento de Formas Compostas:** Em alternativa à deteção das formas geométricas básicas (círculo, quadrado e cruz) implementar um algoritmo que use a detecção dessas em conjunto para garantir a detecção com maior confiança (não detectar por confusão uma base onde não há ao detectar um quadrado no chão que não seja uma base por exemplo) ao usar, por exemplo, uma combinação das detecções das formas com os centros dentro de uma distância pequena o suficiente entre si para confirmar como um conjunto que represente a base (ao escolher essa opção deve-se demonstrar também separadamente o funcionamento da detecção de formas básicas).

  iii. **Medição de Distância:** Estimar em relação à posição da câmera a posição da base detectada. Ou faça isso para um dataset próprio usando as medidas que vpcê projetar para o lado da base e a amplitude da visão da câmera que usar ou, para as fotos fornecidas, use como referência 1m para o tamanho do lado da base e decida algum valor arbitrário para a amplitude do campo de visão da câmera para fins demonstrativos.

  iv. **Comparativo Entre Ferramentas/Algoritmos no Opencv:** Trazer um comparativo de diferentes opções disponíveis (dentre essas básicas usadas com o OpenCV) para implementar alguma etapa (filtro, método de segmentação por cor, método de extração de contorno, método de categorização de contorno etc) do algoritmo, fazer a demonstração de semelhanças e diferenças entre abordagens escolhidas para comparar e explicar na parte escrita algumas diferenças teóricas e dos resultados.

v. **Comparativo com Outros Métodos de Visão Computacional para essa Tarefa:** Trazer o comparativo do uso dos métodos básicos, como eles estão propostos na parte principal da atividade, com métodos de aprendizado de máquina como redes neurais convolucionais usadas em sistemas como o YOLO, para isso deve-se treinar algum modelo de aprendizado de máquina (no sistema da sua escolha), fazer a demonstração comparando ao com OpenCv e explicar na parte escrita algumas diferenças teóricas e dos resultados.

✅ **Checklist de Desenvolvimento:**

- ☐ Configurar o ambiente Python com OpenCV.
- ☐ Estudar a documentação e tutoriais de OpenCV sobre filtros, morfologia, cores e detecção de contornos.
- ☐ Implementar a detecção básica de formas geométricas e cores em imagens estáticas.
- ☐ Adicionar sliders para ajuste de parâmetros de cor.
- ☐ Implementar a visualização das formas detectadas.
- ☐ Implementar pelo menos dois requisitos opcionais.
- ☐ Testar o programa com diferentes imagens e vídeos.
- ☐ Documentar o código e preparar a apresentação.
- ☐ Criar a documentação/relatório do projeto em forma de README.

2️⃣ Opção 2: Criação de Missões Paramétricas em Simulação ROS2/PX4

Objetivo: Desenvolver missões de navegação autônoma configuráveis por parâmetros no ambiente de simulação ROS2/PX4. O projeto visa aprofundar o entendimento de ROS2, PX4 e Gazebo, focando na criação de missões flexíveis e no monitoramento de dados via terminal.

📚 **Materiais de Estudo:**

- **Docker e PX4:**
  - Documentação Docker da PX4 (Recomendado): Instruções para instalar e configurar o ambiente de simulação PX4 com Docker.
- **ROS2 e Integração PX4:**
  - Documentação geral de ROS2 na PX4: Guia de integração ROS2 com PX4, incluindo navegação, offboard control e simulação, navegue dentro dos tópicos na barra lateral para encontrar informações relevantes
  - Exemplo de navegação básica com ROS2 e PX4 em C++: Interessante ler as partes de setup e observar os links para outras partes da documentação da PX4 na página mesmo se for implementar em Python

    - Exemplo de navegação básica com ROS2 e PX4 em Python: Repositório que tem um exemplo que pode ser usado como base da implementação que vá fazer, inclui métodos e atributos que podem ser reutilizados da classe OffboardControl que extende Node
- **ROS2 Tutoriais Gerais:**
    - ROS2 Tutorials: Focar em "Understanding ROS2 nodes", "Topics", "Services", "Actions", "Writing a simple publisher and subscriber", "Using parameters"
    - ROS2 Tutorial em vídeo da implementação de parâmetros para nós: Vídeo sobre um robô terrestre mas que explica bem como usar parâmetros em nós ROS2
    - Tuturial em vídeo de ROS2 com PX4 em Python: Extende e explica o exemplo de navegação básica em Python, inclui dicas importantes e detalhes de instalação e setup do ambiente e de uso do Gazebo
- **Conceitos Fundamentais:**
    - **ROS2 Nodes, Topics, Services, Actions, Parameters:** Compreender a arquitetura ROS2 e como os nós se comunicam usando tópicos, serviços e actions, e como configurar parâmetros em nós ROS2.
    - **Simulação PX4/Gazebo:** Familiarizar-se com o ambiente de simulação PX4 no Gazebo, incluindo a execução de simulações e a interação com o drone simulado.
    - **MAVLINK e PX4_msgs:** Entender as mensagens MAVLINK e PX4_msgs utilizadas para comunicação com o PX4 e como monitorar dados relevantes (e.g., posição, attitude) através de tópicos ROS2.

🎯 **Requisitos Funcionais:**

- **Requisitos Obrigatórios:**
    i. **Simulação ROS2/PX4:** Configurar e executar a simulação de um drone no Gazebo da PX4 integrada ao ROS2.
    ii. **Missões Paramétricas:** Desenvolver um nó ROS2 em Python que permita criar missões de navegação autônoma básicas (decolar, voar para um ponto, pousar) configuráveis através de parâmetros ROS2 (e.g., destino, altura de voo, velocidade). Os parâmetros devem ser definidos no launch file, arquivos de configuração ou via linha de comando no momento da execução.
    iii. **Monitoramento via Terminal:** Ser capaz de ler e interpretar dados de telemetria do drone (posição X, Y, Z) exibidos no terminal em tempo real utilizando `ros2 topic echo`.
    iv. **Documentação da Missão:** Documentar os parâmetros configuráveis da missão, os tópicos ROS2 utilizados para monitoramento e os passos para executar e configurar a simulação.
- **Requisitos Opcionais (Escolher pelo menos 2):**
    i. **Tipos de Trajetória Parametrizáveis:** Expandir as missões para incluir diferentes tipos de

trajetória parametrizáveis (e.g., voo em círculo com raio e centro configuráveis, voo em espiral ou em uma trajetória desenhando um 8, voo em zig-zag, voo para busca em quadrantes etc).

ii. **Variáveis de Controle Parametrizáveis:** Permitir configurar parâmetros de controle da missão (e.g., velocidade, aceleração e ângulo máximos permitidos, taxa de subida/descida, valores de PID etc) via parâmetros ROS2.

iii. **Controle por teclas:** Implementar um controle manual do drone via teclado (e.g., decolagem, pouso, controle de posição) para testes e demonstração interativa.

iv. **Dashboard Web Básico:** Criar um dashboard web básico com algum framework da sua escolha para visualizar os dados de telemetria do drone (posição, altura, setpoint etc) de forma gráfica e interativa.

v. **Testes com Diferentes Parâmetros:** Realizar testes da simulação com diferentes configurações de parâmetros e documentar os resultados e observações sobre o comportamento do drone.

✅ **Checklist de Desenvolvimento:**

- ☐ Instalar Docker e configurar o ambiente com a imagem da PX4 com ROS2 ou instalar os requisitos diretamente na sua máquina.
- ☐ Estudar a documentação de ROS2 e PX4, focando em navegação, simulação e parâmetros.
- ☐ Criar um pacote ROS2 e um nó de controle em Python para missões paramétricas.
- ☐ Implementar missões básicas configuráveis por parâmetros no Gazebo.
- ☐ Monitorar dados de telemetria no terminal usando `ros2 topic echo`.
- ☐ Implementar pelo menos dois requisitos opcionais.
- ☐ Documentar o código, parâmetros e preparar a apresentação.
- ☐ Criar a documentação/relatório do projeto em forma de README.

3️⃣ Opção 3: Planejamento de Missão com Máquina de Estados ou Árvore de Comportamento

Objetivo: Desenvolver um sistema de planejamento de missão para um drone, utilizando Máquina de Estados Finita (FSM) ou Árvore de Comportamento (Behavior Tree - BT) para lógica de decisão e controle. O projeto foca na implementação da lógica de planejamento e na justificativa das escolhas de design, sem a necessidade de simulação em Gazebo (testes automatizados opcionais).

Descrição da missão: O drone deve seguir uma linha predefinida até uma base de pouso identificada por um padrão específico (e.g., cor, forma, código visual). A missão consiste em decolar, seguir a linha, detectar um QR code (com um valor não conhecido previamente mas restrito às

opções) no fim da linha, detectar a base correta para o pouso (a que seja correspondente ao identificador passado pelo QR code dentre outras bases com outros identificadores) e pousar nela de forma segura. O QR code poderá conter o código "1", "2", "3" ou "4", as bases poderão estar na posição "A", "B", "C" ou "D", as posições são conhecidas previamente mas o número correspondente à base localizada não, e a base correta para o pouso será a que tiver o mesmo código numérico que o QR code detectado. O sistema de planejamento deve lidar com diferentes situações como: desalinhamento com a linha (mas não a perda completa dela do campo de visão nem a perda da informação do sentido a seguir na linha), falha ao ler o QR code daquela posição (que pode não acontecer em uma outra tentativa de leitura) e deve tomar decisões baseadas em eventos e condições, o sistema de planejamento não deve ter que lidar com questões como: malhas de controle de baixo nível responsáveis pela estabilidade e afins, implementação de algorítimos e funcionalidades (como visão computacional e geração de trajetórias), deve-se fazer os nós folhas ou condições de transições conceituais, sem necessidade de serem funcionais.

📚 **Materiais de Estudo:**

- **Máquinas de Estados e Árvores de Comportamento:**
  - **Bibliotecas com Implementações:**
    - python-statemachine: Biblioteca Python para Máquinas de Estados Finitas.
      - Exemplos na documentação: Clique nos exemplos para ver o código e detalhes.
      - Github da biblioteca: Tem um exemplo e explicação de como gerar o gráfico da máquina de estados. Mas é preferível usar o PIP para instalar a clonar o repositório
    - py_trees: Biblioteca Python para Árvores de Comportamento.
      - Tutoriais na documentação: Passo à passo da implementação de alguns exemplos.
    - BehaviourTree.cpp: Biblioteca C++ para Árvores de Comportamento, apesar de não ser em Python é uma biblioteca com boas explicações na documentação e com 2 versões de uma excelente ferramenta para desenvolver as árvores de comportamento com uma interface gráfica, fazendo com que seja possível cumprir os requisitos do trabalho com muito pouco ou quase nenhum conhecimento de C++. É uma documentação interessante para a leitura mesmo se for usar outra implementação, traz boas explicações dos conceitos básicos das Árvores de Comportamento.
      - Tutoriais na documentação: Clique nos exemplos para ver o código e detalhes.
      - Github do Groot Open Source: Interface gráfica para criação de árvores de comportamento, mais antiga mas de código aberto e sem recursos pagos.
      - Groot2: Uma interfaçe gráfica mais moderna e com mais recursos, tem todos os recursos necessários para completar a atividade na sua versão gratuita mas

também possiu uma versão paga.

- **Artigos e Tutoriais:**
  - Artigo sobre Árvores de Comportamento em Robótica: O artigo traz, para um escopo intrudutório, um boa explicação do funcionamento e da implementação de uma árvore de comportamento, além de comparar as duas principais implementações `py_trees` e `BehaviorTree.cpp` e uma breve comparação com o uso de Máquinas de Estados Finitos

- **Conceitos Fundamentais:**
  - **Máquinas de Estados Finitas (FSM):** Entender o conceito de estados, transições e eventos em FSMs, e como modelar o comportamento de um sistema usando FSMs.
  - **Árvores de Comportamento (BT):** Entender a estrutura hierárquica de BTs, nós de controle (sequência, seletor), nós de ação e nós de condição, e como usar BTs para planejamento de comportamento complexo.
  - **Planejamento de Missão:** Conceituação de missões autônomas, etapas e estratégias de uma missão típica de drone (decolagem, navegação, pouso), e lógica de decisão para lidar com diferentes situações e eventos durante a missão.

🎯 Requisitos Funcionais:

- **Requisitos Obrigatórios:**
  i. **Implementação de FSM ou BT:** Escolher e implementar um sistema de planejamento de missão usando Máquina de Estados Finita (com `python-statemachine`) **ou** Árvore de Comportamento (com `py_trees` ou `BehaviorTree.cpp`).
  ii. **Representação Visual da Implementação:** Obter um desenho esquemático da estrutura de decisão (FSM ou BT) que a biblioteca de escolha gere com a estrutura pronta que explique a missão (é interessante fazer algum esquemático, à mão ou não durante a implementação do sistema de planejamento projetado mas o que é pedido aqui é o que alguma ferramenta de vizualização da biblioteca ou externa que se associe à ela gere a partir do código).
  iii. **Execução em Ambiente Simulado (Texto/Console):** Usar as ferramentas da biblioteca para mostrar no terminal a execução do caminho desenvolvido pelo sistema de planejamento projetado, pode-se usar a parte visual em tempo real nativa do Groot2 se estiver usando-o, se não demonstre com os prints no terminal que a biblioteca de escolha implemente.
- **Requisitos Opcionais (Escolher pelo menos 2):**
  i. **Simulação de Cenários Diversos:** Desenvolver um ambiente de teste simulado em texto/console para demonstrar o funcionamento da lógica de planejamento a partir da interação com o usuário (pelo terminal mesmo) para a mudança de um estado para algum dos outros

possíveis a cada transição ou finalização, ou não, de um comportamento num dado tick. Na demonstração desse requisito deve-se mostrar a execução de diferentes cenários simulados, inclusive os que não sigam o melhor caminho (e.g., desalinhamento com a linha, falha na leitura do QR code) e a possibilidade de se recuperar deles.

ii. **Testes Automatizados (Cadeia de Markov - Opcional Avançado):** Implementar um sistema de testes automatizados utilizando um modelo de Cadeia de Markov (sistema estocástico) para avaliar a robustez e eficiência do sistema de planejamento em diferentes cenários simulados (e.g., variações no ambiente, falhas de detecção). Esse simulador deve permitir simular os diferentes resultados (tempo para completar, quantidade de tentativas etc) do progresso da missão com diferentes condições (sequência seguida) para completar a missão.

iii. **Justificativa de Escolhas:** Documentar e justificar as escolhas de design e implementação do sistema de planejamento (FSM ou BT), incluindo a estrutura de estados/nós, as transições/fluxo de controle, e o tratamento de eventos e condições.

✅ **Checklist de Desenvolvimento:**

- ☐ Estudar o conceito de Máquinas de Estados Finitas ou Árvores de Comportamento e as bibliotecas correspondentes.
- ☐ Projetar a arquitetura do sistema de planejamento (FSM ou BT) para a missão seguidor de linha.
- ☐ Implementar a FSM ou BT em Python ou C++, utilizando a biblioteca escolhida.
- ☐ Documentar as escolhas de design, a implementação e os testes.
- ☐ Implementar pelo menos dois requisitos opcionais.
- ☐ Preparar a apresentação, focando na explicação da lógica de planejamento e justificativas.
- ☐ Criar a documentação/relatório do projeto em forma de README.

🎯 Avaliação Geral dos Projetos

A avaliação dos projetos de software considerará os seguintes critérios:

1. **Funcionalidade e Cumprimento dos Requisitos:** O projeto atende aos requisitos obrigatórios e opcionais escolhidos? Funciona conforme o esperado?
2. **Qualidade do Código:** O código é bem organizado, legível, comentado e segue boas práticas de programação (quando aplicável à linguagem)?
3. **Documentação e Explicação Técnica:** A documentação (comentários/docstrings no código e o README) é clara e suficiente para entender o projeto? A documentação explica

adequadamente a implementação e a função de cada parte do código? As instruções no README são suficientes para saber quais são os requisitos, dependências e processos para instalar o que for necessário e rodar o código? O candidato demonstra compreensão técnica do projeto durante a apresentação e consegue responder a dúvidas?

4. **Criatividade e Iniciativa (Requisitos Opcionais):** A escolha e implementação dos requisitos opcionais demonstram iniciativa e interesse em expandir o conhecimento sobre a tarefa mas ainda conhecendo as próprias limitações técnicas?
5. **Justificativa de Design (Opção 3):** Para a Opção 3, a justificativa das escolhas de design da Máquina de Estados ou Árvore de Comportamento é clara, lógica e bem fundamentada?

🗂️ Recursos e Referências Adicionais

- ROS2 Documentation: Documentação oficial ROS2.
- PX4 Documentation: Documentação PX4.
- OpenCV Documentation: Documentação oficial OpenCV.
- QGroundControl: Interface gráfica para controle de drones PX4.
- Gazebo: Simulador de robótica 3D.

Boa sorte com o seu projeto! Em caso de dúvidas não hesite em entrar em contato com algum líder de Controle e Sistemas Embarcados.