Projeto de Recrutamento - 1º Semestre 2025/26

ATLAS - *Systems & Aeronautics*

Coordenador Geral do Projeto:
Filipa Fernandes

Autores:

Sofia Gomes	-	Líder de Aerodinâmica
Bruno Marques	-	Co-Líder de Estruturas
Gustavo Toste	-	Co-Líder de Estruturas
João Santos	-	Co-Líder de Estruturas
Lucas Moreira	-	Líder de Sistemas
Dinis Madeira	-	Líder de Controlo
Guilherme Lopes	-	Co-Líder de Software & AI
José Eira	-	Co-Líder de Software & AI
Duarte Pardal	-	Co-Líder de Marketing
Gonçalo Marques	-	Co-Líder de Marketing

Setembro de 2025

7.1 Objetivos

7.1.1 Tracks de Software:

- **Software:** Desenvolver competências em pipelines de captura, processamento e transmissão de vídeo, integração com AI, programação em Python, e domínio da framework Deepstream da Nvidia.
- **SLAM:** Aprender e implementar Visual Odometry e ORB-SLAM3 em ROS2, integrar navegação autónoma de drones com mapeamento e localização em tempo real.
- **Website:** Criar e manter um website dinâmico e interativo para apresentar a equipa, projetos e histórico do ATLAS, incluindo conteúdos multimédia e documentação.

7.2 Tarefas - Track Software

7.2.1 Tarefa 0 - Instalações (Pré-recrutamento, semana 1)

Os recrutas podem realizar as tarefas no sistema operativo que preferirem, desde que este funcione corretamente, sendo, no entanto, recomendada a utilização de Linux, por ser o ambiente de trabalho utilizado pela equipa e aquele em que é possível prestar apoio com maior facilidade (é também o único cuja compatibilidade está garantida, não tendo sido testada a articulação de Gstreamer, OpenCV e outros em Windows ou MacOS). Dito isto, à partida Ubuntu 20 ou 22 funcionará como previsto. Caso o recruta necessite, poderá ser prestado apoio na configuração de Dual Boot. Se for necessário, poderão ser disponibilizadas pens USB e discos externos para salvaguardar dados antes da instalação. Em alternativa, os recrutas podem recorrer a Máquinas Virtuais, caso não pretendam realizar Dual Boot. Adicionalmente, é obrigatório instalar o OpenCV com suporte para Gstreamer.

7.2.2 Tarefa 1 - Introdução aos protocolos de rede e transmissão vídeo (Pré-recrutamento, semana 2)

Esta tarefa tem como objetivo familiarizar os pré-recrutas com os protocolos necessários para estabelecer uma pipeline de transmissão de vídeo. Para tal, os candidatos deverão elaborar um relatório breve sobre os tópicos abaixo indicados, podendo acrescentar outros, caso considerem relevante. O relatório poderá ser complementado com elementos adicionais que auxiliem a clarificar a compreensão do tema, como diagramas e esquemas. Poderá ainda incluir pequenos testes com os protocolos, cujos resultados podem ser apresentados através de screenshots, screencasts ou vídeos. Sempre que os testes exijam mais elementos além de comandos no terminal para serem reproduzidos (por exemplo, scripts), estes deverão ser igualmente partilhados. Seguem-se os tópicos:

1. DHCP, DNS, ARP, IP address, MAC address, subnetmask, default gateway: o que são, para que servem e como se relacionam
2. RTSP, RTP, TCP, UDP, ports: o que são, para que servem e como se relacionam
3. Codec, parser, bitrate, jitter buffer latency: qual o seu papel numa stream
4. Gstreamer, OpenCV e ROS2: o que são, para que servem, comandos e funções básicas

Nota: os subtópicos dentro de cada tópico 1 a 3 são relativamente simples, pelo que não é necessário despender demasiado tempo com o texto de cada um. O foco deve incidir sobretudo nas relações entre os diferentes subtópicos. Embora, em última análise, todos os conceitos estejam interligados, o objetivo é que os candidatos se concentrem nas ligações mais diretas. A realização de testes e scripts não é obrigatória, mas poderá revelar-se útil para uma melhor preparação da tarefa seguinte. Caso o entendam pertinente, os candidatos podem solicitar feedback relativamente a esses elementos.

7.2.3 Tarefa 2 - Exercício prático e projeto (Pré-recrutamento, semanas 3 a 7)

Esta tarefa tem como objetivo consolidar a aprendizagem da última tarefa.

Projeto: Os pré-recrutas irão realizar um projeto individual, onde terão de aplicar os conhecimentos obtidos. O código será avaliado pelos colíderes, com os critérios habituais de um projeto de programação. O objetivo do projeto é a construção de uma pipeline simples de captura e display de vídeo, em Python, com recurso ao Gstreamer e ao OpenCV. Deve ser elaborado por etapas:

1. Captura de vídeo a partir da webcam com Gstreamer (versão pelo terminal e script em python).
2. Processamento de video com OpenCV: inverter vídeo recebido, aplicar saturação, ou qualquer outra coisa (fica ao vosso critério, escolham o que gostarem mais)

   Nota: para testar as duas etapas seguintes, caso os candidatos não disponham de dois computadores, poderá ser útil recorrer a uma Virtual Machine. Sempre que possível, os testes poderão também ser realizados com recurso a hardware do ATLAS, sob supervisão dos colíderes.
3. Redirecionamento de vídeo: em vez de abrir o vídeo no computador de origem, este deve ser capturado, processado e enviado para outro computador, onde será reproduzido através de um comando do Gstreamer no terminal (ou, em alternativa, através de um script em Python que aguarde a receção do vídeo).
4. Captura de vídeo a partir de uma stream rtsp: ligação de uma câmara do ATLAS ao computador do candidato, ou, em alternativa, utilização de outro computador para enviar vídeo por RTSP, simulando a câmara. Nesta fase devem estar envolvidos três intervenientes: a câmara (ou câmara simulada), o computador que processa o vídeo e o computador que o recebe.
5. Captura de vídeo a partir de duas streams rtsp distintas: ambas as streams devem ser processadas (por exemplo, uma invertida e outra com saturação aumentada) e enviadas separadamente para o computador de destino, que deverá reproduzir os dois fluxos em janelas independentes.
6. ROS2: publisher node no computador que processa e envia o vídeo, que publica uma mensagem (basta um hello world) para um tópico; listener node no computador que abre o vídeo que subscreve a este tópico.
7. Bónus: utilizar a framework Deepstream da Nvidia
8. Bónus: refazer etapas em C++

Nota: Os recrutas devem submeter o código do projeto acompanhado de pequenos vídeos de demonstração. Os vídeos não têm de estar editados; basta mostrar que funciona.

Serão dadas mais informações na altura (bibliotecas que devem instalar, versão do Python a utilizar, etc). Pode vir a ser acrescentada uma integração com AI (processar o vídeo com o modelo de AI antes de dar display, em vez de inverter/saturar/outro). As etapas bónus não têm de ser realizadas por nenhuma ordem. Aliás, não têm de ser realizadas de todo.

Dica: é natural que os recrutas tenham dúvidas ao longo do processo. Os LLMs apresentam, por vezes, limitações na geração de código, sobretudo quando este envolve interação com hardware, como câmaras ou computadores de bordo, o que pode dificultar a aprendizagem e atrasar o progresso no projeto. Assim, recomenda-se que os candidatos procurem apoio junto dos líderes de equipa sempre que necessário. O objetivo não é que dominem todos os conteúdos de imediato, mas que evoluam com acompanhamento adequado.

7.2.4 Tarefa 3 - Introdução às pipelines de transmissão de vídeo no ATLAS (recrutamento, semana 9)

Esta tarefa tem como objetivo familiarizar os recrutas com as pipelines de tranmissão de vídeo utilizadas pela equipa. Irão aprender sobre as pipelines destinadas ao VTOL e ao mini-drone, garantindo que compreendem na totalidade os seguintes conceitos:

1. Threads, paralelismo, buffers
2. Modelos de AI para Computer Vision e ferramentas utilizadas nos mesmos (YOLO, ZoeDepth, SAHI)
3. Transmissão através de módulos Wi-Fi: wifi broadcast next generation, Ground Station key e Drone key

7.2.5 Tarefa 4 - Otimização da pipeline do VTOL com Deepstream (recrutamento, semana 10 a 13)

Esta tarefa será a primeira contribuição real dos recrutas para o projeto. Irão otimizar a pipelines atuais, com recurso ao Deepstream, uma framework desenvolvida pela Nvidia. O objetivo é criar pipelines com latências mais baixas, capazes de receber streams de fontes diferentes, processá-las em paralelo, e enviar os diferentes vídeos para a Ground Station. Está previstas as seguintes etapas:

1. Dual boot com Linux e instalação de dependências (se não tiver feito no pré-recrutamento)
2. Pesquisa sobre o Deepstream e estruturação da pipeline
3. Implementação do Deepstream
4. Testes (1- laboratório; 2- Jetson; 3- avião 3D)

7.3 Tarefas - Track SLAM

7.3.1 Requisitos Técnicos - recomendações

- **Software**: Ubuntu 20.04/22.04, ROS2 Humble, OpenCV 4.x, Python 3.8+ (ao vosso critério)
- **Câmera**: Câmera stereo USB ou webcam (para testes iniciais; asseguramos as condições para esses testes caso não disponham)
- **Hardware**: 8GB RAM, GPU com CUDA (recomendado), CPU dual-core (apenas necessário para testes em fases mais avançadas; nós asseguramos as condições para esses testes caso não disponham)

7.3.2 Tarefa 1 - Fundamentos de SLAM e Visual Odometry (Semanas 1-2)

Objetivo: Introduzir os conceitos básicos de SLAM através da implementação de Visual Odometry simples e configuração inicial do ORB-SLAM3. Os recrutas devem entregar um relatório breve sobre a tarefa, com explicação dos conceitos teóricos e comparação de resultados, mas o foco é ter o código a correr como previsto.

7.3.3 Semana 1: Teoria e Implementação Básica

1. **Conceitos teóricos**
   - Ler documentação sobre mapping, localization, e loop closure
   - Compreender modelos de câmara pinhole e distorção
   - Estudar transformações entre frames: camera → world → base_link
2. **Implementação de Visual Odometry**
   - Criar nó ROS2 que subscreve `sensor_msgs/Image` de câmara stereo
   - Implementar deteção de features com `cv2.ORB_create()`
   - Fazer matching entre frames consecutivos com `cv2.BFMatcher()`
   - Calcular essential matrix com `cv2.findEssentialMat()`
   - Recuperar pose com `cv2.recoverPose()`
   - Publicar trajetória estimada em `/odom` para RViz

Código exemplo:

```
class BasicVO:
    def __init__(self):
        self.detector = cv2.ORB_create(nfeatures=1000)
        self.matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
        self.pose_pub = self.create_publisher(PoseStamped, '/vo_pose', 10)

    def process_frames(self, img1, img2):
        # Detectar e fazer match de features
        kp1, desc1 = self.detector.detectAndCompute(img1, None)
        kp2, desc2 = self.detector.detectAndCompute(img2, None)
        matches = self.matcher.match(desc1, desc2)

        # Calcular pose change
        E, mask = cv2.findEssentialMat(pts1, pts2, self.camera_matrix)
        _, R, t, mask = cv2.recoverPose(E, pts1, pts2, self.camera_matrix)

        return R, t
```

7.3.4 Semana 2: Instalação e Teste do ORB-SLAM3

1. **Instalação do ORB-SLAM3**
   - Clonar repositório: `git clone https://github.com/UZ-SLAMLab/ORB_SLAM3.git`
   - Instalar dependências: Eigen3, Pangolin, OpenCV
   - Compilar ORB-SLAM3 seguindo instruções oficiais
   - Instalar wrapper ROS2: `git clone https://github.com/zang09/ORB_SLAM3_ROS2.git`
2. **Configuração inicial**
   - Fazer download do vocabulary ORB: `ORBvoc.txt`
   - Criar ficheiro de calibração da câmara (formato YAML)
   - Testar com dataset EuRoC MH01: `rosbag2 play MH01_easy`
   - Verificar output de pose e mapa em RViz
3. **Breve comparação de resultados**
   - Comparar trajetória do Visual Odometry básico com ORB-SLAM3
   - Registar diferenças observadas (drift, precisão, robustez)

Resultado: ORB-SLAM3 inicializa e track corretamente com dataset EuRoC; Visual Odometry básico estima movimento (mesmo com drift).

7.3.5 Tarefa 2 - Integração do ORB-SLAM3 em ROS2 (Semanas 3-4)

Objetivo: Configurar ORB-SLAM3 para funcionar como componente ROS2, publicando poses e mapas em tópicos standard. Submissão do código.

7.3.6 Semana 3: Configuração de Parâmetros e Coordenadas

1. **Calibração de câmara**
   - Usar `ros2 run camera_calibration cameracalibrator.py` para obter parâmetros intrínsecos
   - Criar ficheiro YAML de calibração no formato ORB-SLAM3
2. **Configuração de frames ROS2**
   - Configurar TF tree: `map → odom → base_link → camera_link`
   - Implementar TF broadcaster para publicar transformações
   - Verificar frames com `ros2 run tf2_tools view_frames`
3. **Ajuste de parâmetros ORB-SLAM3**
   - Ajustar `nFeatures`, `scaleFactor`, `nLevels` para o hardware
   - Configurar thresholds de tracking e mapping
   - Testar diferentes configurações com dataset

7.3.7 Semana 4: Publishers ROS2 e Validação

1. **Implementação de publishers**
   - Publicar poses 3D em `/odom` (`geometry_msgs/PoseStamped`)
   - Publicar mapa 3D como `sensor_msgs/PointCloud2` para visualização
   - Garantir timestamps e frame_ids corretos
2. **Tratamento de falhas**
   - Detectar tracking loss através de callbacks do ORB-SLAM3
   - Implementar reinicialização automática em caso de falha
   - Criar service `std_srvs/Trigger` para reset manual
   - Monitorizar estado do sistema (publishing rate, tamanho do mapa)
3. **Validação com datasets**
   - Testar com datasets EuRoC MH01, MH02, MH03
   - Medir precisão com ATE (Absolute Trajectory Error)
   - Avaliar drift relativo à distância percorrida (%)
   - Registar métricas de performance (CPU, memória, FPS)

Resultado: Sistema publica poses 3D de forma consistente, tracking não falha durante datasets completos, e mapa 3D é gerado corretamente.

7.3.8 Tarefa 3 - Gestão de Mapas e Robustez (Semanas 5-6)

Objetivo: Implementar persistência de mapas, multi-sessão, e otimização de performance para uso em tempo real. Submissão do código.

7.3.9 Semana 5: Persistência e Multi-sessão

1. **Configuração de save/load de mapas**
   - Configurar ORB-SLAM3 para usar modo "SaveMap"
   - Testar a funcionalidade de save/load com comandos do ORB-SLAM3
   - Testar save/load de mapas via ROS2 service wrapper
   - Verificar que o sistema consegue relocalizar corretamente após carregar o mapa
2. **Relocalização após perda de tracking**
   - Configurar ORB-SLAM3 para relocalização automática após tracking loss
   - Testar cenários de "robot kidnapping"(mover o robot subitamente para outra posição)
   - Verificar que o sistema consegue relocalizar no mapa existente
   - Medir tempo de relocalização

7.3.10 Semana 6: Otimização de Performance

1. **Profiling e otimização**
   - Usar `htop`, `nvidia-smi`, `ros2 topic hz` para medir performance
   - Identificar bottlenecks (CPU, GPU, RAM, disk I/O)
2. **Monitorização do sistema**

- Criar nó de monitorização que publica system health
- Implementar alertas para: tracking loss, low FPS, high memory usage
- Logging com diferentes níveis (DEBUG, INFO, WARN, ERROR)

Resultados: Sistema executa >20 minutos sem crashes, relocalization funciona consistentemente, performance mantém-se estável.

7.3.11 Tarefa 4 - Integração Completa e Documentação (Semanas 7-8)

Objetivo: Consolidar sistema numa framework robusta com configuração clara e documentação completa. Submissão do código e documentação.

7.3.12 Semana 7: Sistema Unificado

Deliverables:

1. **Launch files estruturados**
   - `slam_bringup.launch.py`: sistema completo com todos os nós
   - `slam_dataset.launch.py`: replay de datasets com ground truth
   - `slam_live.launch.py`: câmara ao vivo com visualização
   - Organizar parâmetros em ficheiros YAML separados por componente
2. **Ferramentas de debug**
   - Configurar RViz com displays para: pose, map, features, trajectory
   - Criar scripts de análise: trajectory comparison, map quality metrics
   - Implementar debug topics: matched features, tracking status, timing info
3. **Configuração multi-ambiente**
   - Parâmetros otimizados para: indoor, outdoor, low-light
   - Diferentes configurações de câmara: monocular, stereo, RGBD
   - Templates de calibração para câmaras do ATLAS

7.3.13 Semana 8: Testes e Documentação

1. **Testes sistemáticos**
   - Testar em Gazebo worlds
   - Validar com diferentes condições: day/night, textured/textureless surfaces
   - Performance benchmarks: initialization time, tracking accuracy, map size
   - Criar scripts de teste automatizados
2. **Documentação completa**
   - README com installation guide step-by-step
   - Configuration guide: camera calibration, parameter tuning
   - Troubleshooting guide para problemas comuns

Resultados: Sistema funciona seguindo a documentação, diferentes utilizadores conseguem replicar setup, troubleshooting guide resolve problemas comuns.

7.3.14 Tarefa 5 - Navegação Autónoma de Drone com SLAM (Semanas 9-12)

Objetivo: Integrar o SLAM com controladores do drone para navegação autónoma 3D, usando as poses do ORB-SLAM3 como referência para controlo e planeamento de trajectórias. Submissão do código e documentação.

Nota: Várias partes desta tarefa incluem trabalho da equipa de Controlo. O que já estiver feito, ou que a equipa de Controlo tenha disponibilidade para fazer, deve ser aproveitado, aliviando trabalho aos recrutas. Caso contrário, os recrutas terão de fazer, para poderem avançar na tarefa. Nesse caso, não é esperado que façam um trabalho espetacular nas partes de controlo, apenas o necessário para continuarem.

7.3.15 Semana 9-10: Planeamento e Seguimento de Waypoints 3D

1. **Implementação de nó de waypoint following**
   - Criar nó ROS2 que recebe lista de waypoints 3D
   - Implementar controlo de posição, roll, pitch, yaw, velocidade etc do drone com poses do ORB-SLAM3
2. **Simulação de trajetórias**
   - Testar seguimento de trajectórias simples em Gazebo
   - Visualizar trajectória no RViz usando pose do ORB-SLAM3

7.3.16 Semana 11-12: Missão Autónoma e Validação

1. **Missão de demonstração**
   - Criar cenário 3D com obstáculos e waypoints distribuídos
   - Implementar missão completa: takeoff → map → navigate → land
   - Gravar vídeo de demonstração
   - Medir métricas de desempenho: tempo total, sucesso nos waypoints, precisão
2. **Documentação de integração**
   - Guia para setup do ORB-SLAM3
   - Limitações e workarounds encontrados

Resultados: Drone navega autonomamente usando poses do SLAM, completa missão 3D com waypoints e obstáculos, e sistema está documentado para replicação futura.

7.4 Tarefas - Track Website

7.4.1 Tarefa 1 - Estrutura Inicial do Website (Semanas 1-2)

Objetivo: Criar a base do website do ATLAS, reproduzindo funcionalidades da webpage do ATLAS no site Aerotec, mas com páginas separadas para cada equipa.

1. Criar layout global: header, footer, barra de navegação
2. Criar páginas para cada equipa atual
3. Criar página “Sobre o ATLAS” e “Equipa anterior"(nesta tarefa basta um esboço, para mais tarde colocar os membros do ATLAS em 24-25; depois, adicionar 23-24, etc)
4. Configurar routing (React Router / equivalente)
5. Testes básicos de responsividade e navegadores

7.4.2 Tarefa 2 - Página de Equipas e Conteúdo Dinâmico (Semanas 3-4)

Objetivo: Criar páginas dinâmicas para as equipas

1. Criar componente genérico para apresentar cada equipa (foto, descrição, membros)
2. Popular informação via JSON ou API local
3. Adicionar animações leves e transições para melhor UX

7.4.3 Tarefa 3 - Conteúdos e Funcionalidades Adicionais (Semanas 5-6)

Objetivo: Tornar o website mais interativo

1. Galeria de imagens e vídeos de projetos
2. Seção de notícias ou atualizações (feed simples)
3. Mapa interativo do laboratório, de locais das competições, etc com recurso a fotografias nossas e APIs do Google Maps, Leaflet ou Mapbox
4. Localização do ATLAS (laboratório, local das competições)
5. Pequenos efeitos de interação: hover effects, scroll animations

7.4.4 Tarefa 4 - Testes, Performance e Documentação (Semanas 7-8)

1. Testar website em diferentes browsers e dispositivos
2. Otimização de imagens e recursos
3. Verificar carregamento rápido e responsividade
4. Documentar setup, deployment e manutenção

7.4.5 Tarefa 5 - Deployment e Finalização (Semanas 9-12)

1. Configurar hosting do website
2. Garantir URLs amigáveis e routing correto
3. Testes finais com conteúdo das equipas
4. Manual de utilização para futuras atualizações do site

Resultados: Website funcional, responsivo, com páginas separadas por equipa, histórico de anos anteriores, conteúdo multimédia, e documentação completa para manutenção e atualização.