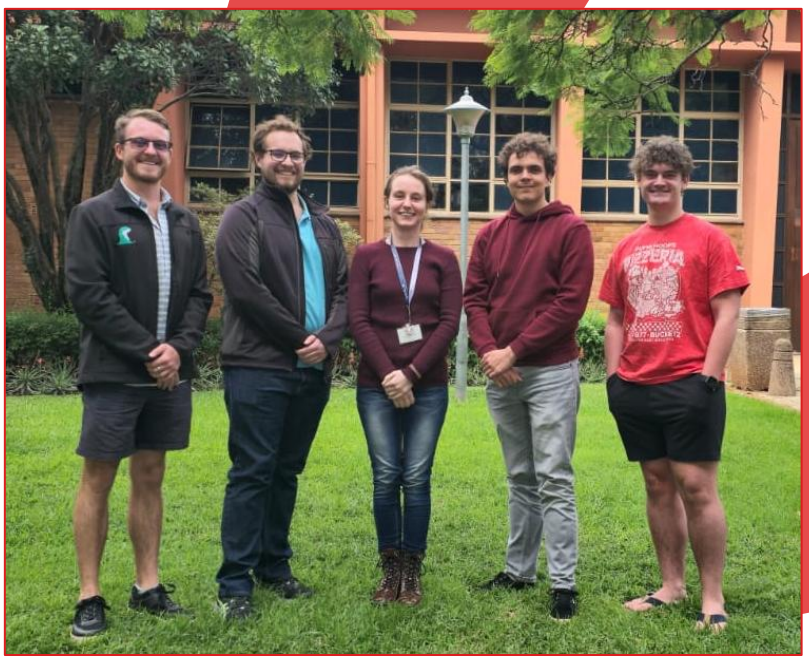
SuperLap Racing Line Optimization System

EPI-USE



Quintessential

Amber Ann Werner [u21457752]

Milan Kruger [u04948123]

Qwinton Knocklein [u21669849]

Sean van der Merwe [u22583387]

Simon van der Merwe [u04576617]

Coding Practices

- **Naming**: Functions and files are named to clearly reflect their purpose or output. Descriptive naming is prioritized over name length limitations.
- **Structure**: Code is kept modular and functions are designed to handle specific tasks where possible.
- **General Practices**: Standard coding practices are followed, including avoiding deeply nested logic, keeping code readable, and minimizing redundancy.

Version Control Guidelines

Commit Messages: All commit messages must be clear, descriptive, and explain what the commit does.

Branching Strategy:

The primary branches are:

- `main`: Stable production-ready code.
- `dev`: Integration branch for completed features.

Feature branches are categorized by function:

- `UI/`: Frontend and website-related work
- `Backend/`: Backend processing and API
- `CICD/`: Continuous Integration and Deployment scripts/tests

Branch naming follows a consistent format:

- Example: Backend-PSA-start, UI-Web-LandingPage

Commit Frequency: Developers are expected to make a minimum of 10 commits per week, ideally after every significant update on their feature branch.

Pull Requests:

- Pull requests must be submitted once a branch feature is complete.
- Each PR must be reviewed by **at least two team members** before being merged.

- Branches are merged progressively: `feature → category (e.g: UI) → dev → main`.
- Direct commits to main are not allowed.

CI/CD: The main branch runs the CI/CD pipelines to ensure stability.

Tools and Configurations

CI/CD: A basic CI/CD setup is implemented, currently running automated tests from the various tests folders.

Docker:

- Each backend component (API, ImageProcessor, RacelineOptimizer) has its own `Dockerfile`.
- A root-level `docker-compose.yml` is used to orchestrate the containers.

Scripts: Utility scripts are stored in the scripts/ directory for local tool setup and CI/CD helpers.

Linters/Formatters: Not strictly enforced, but individual team members may use personal formatting tools suited to their language. There is also currently linting present in our C# code.

Language/ Framework-Specific Conventions

Unity and RacelineOptimizer: Written in C#. Follows typical Unity/C# naming and structure conventions.

API: Implemented in Node.js using JavaScript/TypeScript.

Image Processor: Written in Python, using common Pythonic conventions (e.g: `snake_case`, modular scripts).

Website: Built with standard HTML, CSS, and JavaScript, organized within the docs/ folder for GitHub Pages compatibility.