Rusty Autopilot

ECE Capstone

Sponsored by: Adam Wick and Michal Podhradsky, Galois Inc., galois.com

Background and Motivation

Rust is a modern, memory safe language, intended to replace C/C++ in various applications. One large obstacle preventing Rust from becoming more popular is the fact that large legacy codebases are written in C, and transcribing C into Rust by hand can be a difficult, time-consuming, and error-prone experience. To tackle this problem, Galois worked on an automated tool C2Rust - a tool that lets you automatically translate C code to unsafe Rust.

We want to show how C2Rust can be useful for embedded applications., by showing how we can start with the Cleanflight autopilot for racing drones, written in C/C++, and transforming it into a well-structured and flexible Rust implementation: *Rustyflight*. Cleanflight is largely self-contained – it doesn't depend on large external libraries or board support packages – and so represents something of an ideal case for these sorts of projects. That being said, what is needed is a mechanism to perform the translation, address several build system concerns, and then generate an evaluation suite that ensures continual correctness as the code is gradually refactored. This is where the capstone team steps in.

Objectives

Currently, we are facing these main challenges and we expect the team to address them:

1. **The creation of a high-quality test harness**. Cleanflight provides some unit tests, typically focused on testing particular functions or functional blocks (such as PID controller). For more complex testing scenarios, a Software-In-The-Loop (SITL) target can be used. We expect the team to provide easy to use scripts for testing the generated Rust code; first, to ensure that an initial C2Rust translation preserved the correctness of the original system, and second, to ensure that further refactorings maintain correctness.
2. **Support multiple build configurations**. Cleanflight supports multiple boards, and various configurations. This is managed by a multitude of "ifdef" directives and Makefile flags. C2Rust works with pre-processed source code, where all unused macros and directives are removed. The resulting Rust code hence represents only a singular configuration, which is not ideal. To make Rustyflight more broadly useful, we expect the team to investigate mechanisms to replicate these build alternatives

within the Rust code: as Rust/Cargo feature flags, hierarchical Rust libraries that can be appropriately reassembled, or some alternative solution.

3. **Perform a real flight validation**. The last step is to conduct one or more flights with a racing drone and Rustyflight - the drone will be provided by Galois, and will be similar to Eachine Wizzard x220 with an F3 6DOF Flight controller.

Learning Outcomes

The team will gain expertise in the following:

- Embedded systems development and debugging
- Rust language and cross-compilation
- Testing and validation of automatically generated code

Milestones

The project consists of the following milestones:

- develop a Docker-based environment for customizing and building Rustyflight
- develop testing scripts to validate Rustyflight against the Cleanflight's unit tests
- develop SITL scenarios for Rustyflight that demonstrate all basic functionalities (stabilization and control, sensor fusion, etc.)
- conduct (with Galois' help if needed) a validation flight with the racing drone and Rustyflight

CleanFlight and C2Rust are open source code bases, and we expect Rustyflight to also be an open source project. Galois will provide necessary hardware.