

Retrospective Sprint V

Team 04



Sprint V Goal

Establish robust **Unit Testing (including Code Coverage and Static Analysis)** for the core STM32 firmware logic, and validate the performance of the data pipeline by modeling and measuring latency through the **KUKSA Vehicle Signal Specification (VSS) Data Broker**.

OSS VERIFICATION TOOLCHAIN STRATEGY

Why Open-Source Over Commercial Solutions?



CORE STRATEGY

COST CONTROL	TECHNICAL AGILITY	VENDOR INDEPENDENCE	NATIVE EFFICIENCY
No recurring licensing fees, Maximize resource allocation	Modular, Flexible integration into Linux / Yocto (AGL)	Long-term control over entire toolchain stack	Language-native tools, Minimize external dependencies

Commercial Tools Discarded: VectorCAST, Parasoft, Helix QAC, Polyspace

OSS VERIFICATION TOOLCHAIN STRATEGY

A

Unit Testing Framework & Mocking

Unity + CMock

Pure ANSI C, Minimal footprint, Auto-generated mocks for ThreadX RTOS

cargo test

Native Cargo integration, Deterministic host execution



Why NOT Google Test (GTest)?

C++ runtime overhead unsuitable for pure C embedded

B

Static Analysis & Security (SAST)



CodeQL (Core)

Deep flow-of-data analysis, MISRA C/C++, AUTOSAR C++14, Formal compliance gating



Clang-Tidy + Clippy (Complementary)

Developer-facing quality checks, Clang-Tidy for C/C++, Clippy for Rust idioms

Unified Standards Enforcement — ISO 26262 compliance

C

Code Coverage & Quality Metrics

gcov / lcov (C/C++) and grcov (Rust): Branch Coverage → MC/DC (ASIL-D), Unified in SonarQube



MC/DC progression for ASIL-D compliance

OSS VERIFICATION TOOLCHAIN STRATEGY

LCOV - code coverage report				
Current view: top level Test: DrivaPI - Unified Test Coverage Date: 2025-12-19 09:41:31				
	Hit	Total	Coverage	
	Lines:	230	230	100.0 %
	Functions:	34	34	100.0 %
Directory				
/home/sheila/Desktop/drivapi/tests/unit/speed_sensor/src	Line Coverage ↴	100.0 %	37 / 37	100.0 %
src	Functions ↴	100.0 %	193 / 193	100.0 %
	32 / 32			
Generated by: LCOV version 1.14				
LCOV - code coverage report				
Current view: top level - /home/sheila/Desktop/drivapi/tests/unit/speed_sensor/src Test: DrivaPI - Unified Test Coverage Date: 2025-12-19 10:39:38				
	Hit	Total	Coverage	
	Lines:	37	37	100.0 %
	Functions:	2	2	100.0 %
Filename				
speed_sensor.c	Line Coverage ↴	100.0 %	37 / 37	100.0 %
	Functions ↴	2 / 2		
Generated by: LCOV version 1.14				
LCOV - code coverage report				
Current view: top level - src Test: DrivaPI - Unified Test Coverage Date: 2025-12-19 10:39:38				
	Hit	Total	Coverage	
	Lines:	193	193	100.0 %
	Functions:	32	32	100.0 %
Filename				
dc_motor_functions.c	Line Coverage ↴	100.0 %	29 / 29	100.0 %
dc_motor_testable.c	Functions ↴	8 / 8		8 / 8
pcf8574 testable.c	100.0 %	100 / 100	100.0 %	8 / 8
servo_functions.c	100.0 %	22 / 22	100.0 %	1 / 1
servo_motor_testable.c	100.0 %	12 / 12	100.0 %	6 / 6
speed_sensor_functions.c	100.0 %	22 / 22	100.0 %	2 / 2
	7 / 7			
Generated by: LCOV version 1.14				

OSS VERIFICATION TOOLCHAIN STRATEGY

Unit Tests - CMake/Ceedling
feat: Create CI workflow for unit tests #16

Summary

Triggered via pull request 5 minutes ago Status: Success Total duration: 1m 46s Artifacts: -

Jobs: Build and Run Unit Tests

Run details: Usage: Workflow file

unit_tests.yml
on: pull_request

Workflow file: Build and Run Unit Tests 1m 40s

Build and Run Unit Tests summary

Coverage Summary

lines..... 100.0% (230 of 230 lines)
functions.. 100.0% (34 of 34 functions)
branches... no data found

Coverage report generated

Unit Test Results

Unit Test Results

Overall Status: PASSED

- Motor Servo Tests: PASSED (160 tests)
- Speed Sensor Tests: PASSED (11 tests)

Coverage

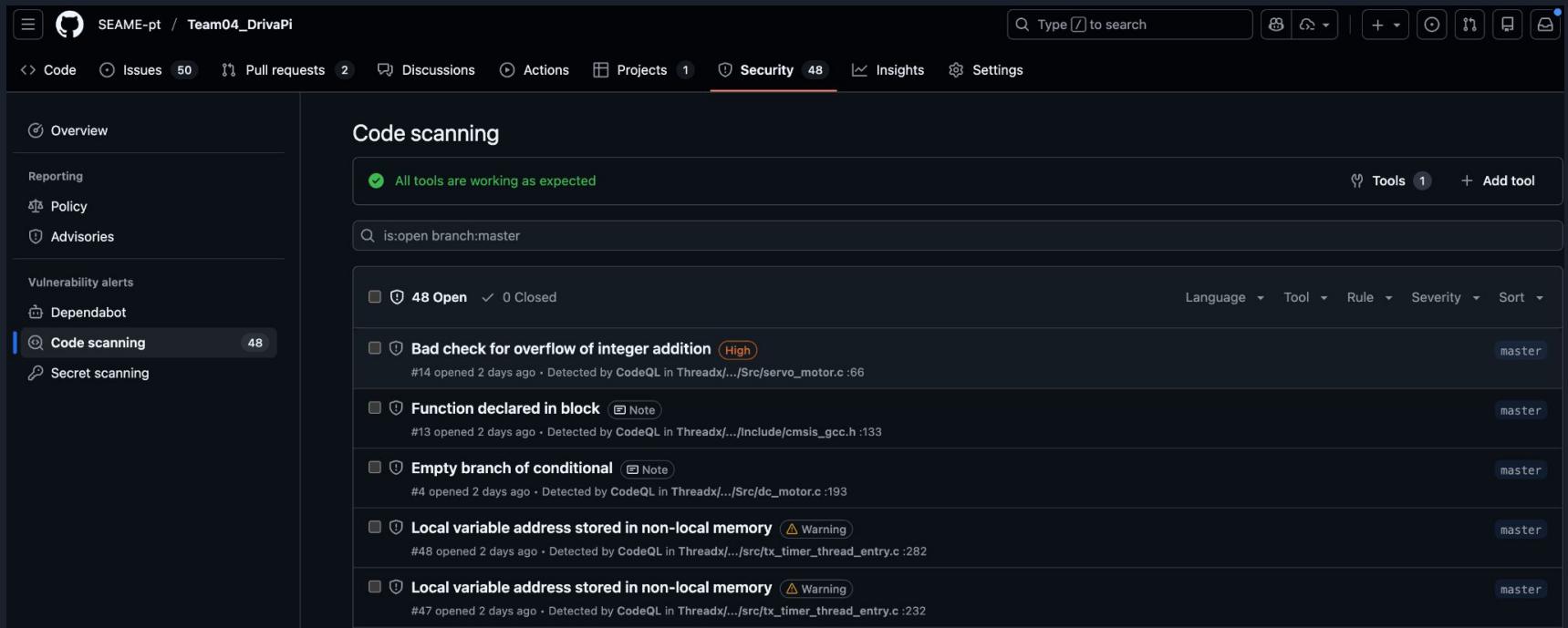
- Lines: 100.0%
- Functions: 100.0%

Full coverage reports available in workflow artifacts

Job summary generated at run-time

OSS VERIFICATION TOOLCHAIN STRATEGY

CODEQL GITHUBACTION



The screenshot shows a GitHub repository interface for the team SEAME-pt / Team04_DrivaPi. The navigation bar includes links for Code, Issues (50), Pull requests (2), Discussions, Actions, Projects (1), Security (48), Insights, and Settings. The Security tab is currently selected.

The main content area is titled "Code scanning". A green checkmark indicates "All tools are working as expected". A search bar contains the query "is:open branch:master".

The "Code scanning" section displays a list of 48 open issues:

- Bad check for overflow of integer addition** [High] #14 opened 2 days ago · Detected by CodeQL in Threadx/.../Src/servo_motor.c :66 · master
- Function declared in block** [Note] #13 opened 2 days ago · Detected by CodeQL in Threadx/.../include/cmsis_gcc.h :133 · master
- Empty branch of conditional** [Note] #4 opened 2 days ago · Detected by CodeQL in Threadx/.../Src/dc_motor.c :193 · master
- Local variable address stored in non-local memory** [Warning] #48 opened 2 days ago · Detected by CodeQL in Threadx/.../src/tx_timer_thread_entry.c :282 · master
- Local variable address stored in non-local memory** [Warning] #47 opened 2 days ago · Detected by CodeQL in Threadx/.../src/tx_timer_thread_entry.c :232 · master

On the left sidebar, the "Code scanning" tab is highlighted with a blue bar, showing 48 issues. Other tabs include Overview, Reporting, Policy, and Advisory.



OSS VERIFICATION TOOLCHAIN STRATEGY

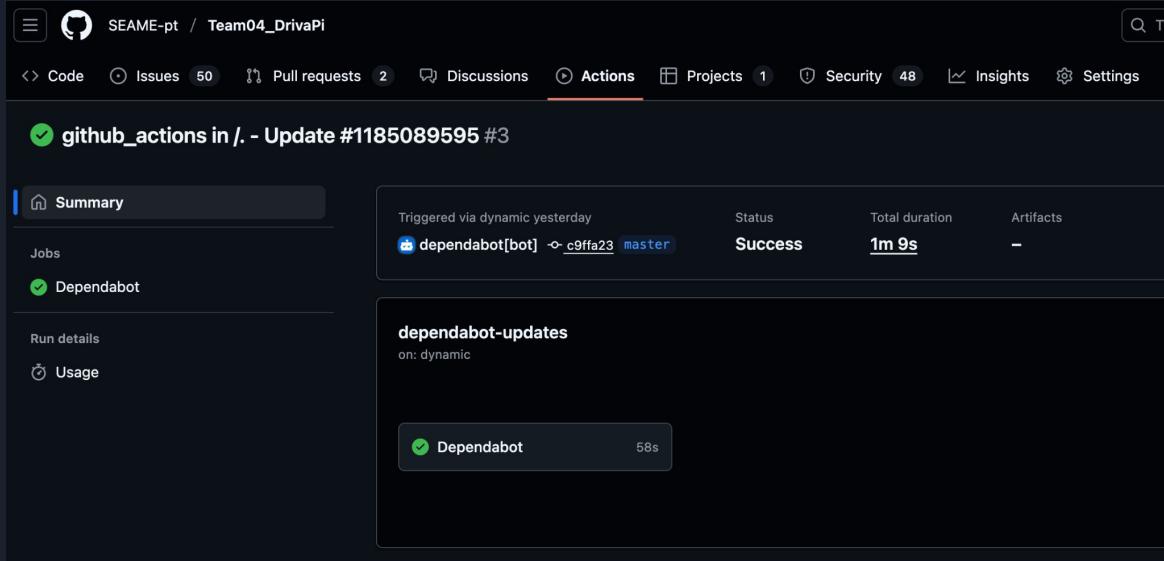
CODEQL GITHUBACTION

The screenshot shows the GitHub interface for the repository "SEAME-pt / Team04_DrivaPi". The top navigation bar includes links for Code, Issues (43), Pull requests (2), Discussions, Actions, Projects (1), Security (2), Insights, and Settings. The Security tab is currently selected, indicated by an orange underline. On the left, a sidebar menu lists Overview, Reporting, Policy, Advisories, Vulnerability alerts, Dependabot, Code scanning (2), and Secret scanning. The main content area is titled "Code scanning" and displays the following information:

- A green checkmark icon indicates "All tools are working as expected".
- A search bar contains the query "is:open branch:master".
- Tool selection: A dropdown shows "Tools 1" with an option to "+ Add tool".
- Status summary: 2 Open issues and 46 Closed issues.
- Issue details:
 - Commented-out code**: An issue labeled "#35" was opened 3 days ago and detected by CodeQL in Threadx.../Src/system_stm32u5xx.c :137. It is marked as "Commented-out code".
 - Long switch case**: An issue labeled "#9" was opened 3 days ago and detected by CodeQL in Threadx.../Src/system_stm32u5xx.c :294. It is marked as "Long switch case".
- Filtering options: Language, Tool, Rule, Severity, and Sort dropdowns.
- Branch filter: A "master" button.

DEPENDENCIES CHECK

Dependabot for Repo's dependencies



The screenshot shows a GitHub Actions run for the repository SEAME-pt / Team04_DrivaPi. The run is triggered via dynamic yesterday and is labeled "dependabot[bot] -> c9ffa23 master". The status is Success, and the total duration is 1m 9s. The run details show a single job named "dependabot-updates" triggered on "dynamic". The job was completed by Dependabot 58 seconds ago.

github_actions in ./ - Update #1185089595 #3

Triggered via dynamic yesterday

dependabot[bot] -> c9ffa23 master

Status: Success

Total duration: 1m 9s

Artifacts: -

dependabot-updates

on: dynamic

Dependabot 58s

QEMU SPIKE

QEMU does not support the STM32U5 (B-U585I-IOT02A) because there is no emulated machine or peripherals for this board.

It can run ARM code, but it cannot model the board's actual hardware.

CAN BUS LATENCY ANALYSIS

Round-Trip
Median

789 µs

One-Way Est.

395 µs

STM32 Internal

248 µs

CAN BUS LATENCY ANALYSIS

Performance Across Three Test Configurations

Configuration	Min	Mean	Median	Max	Std Dev	Jitter	Determinism
STM32 ↔ RPi	710	1 003.47	789	16 724	1 456.21	16 014	 Linux scheduling
RPi Loopback	522	592.37	593	721	18.96	199	 Good
STM32 Internal	244	247.77	248	256	2.08	12	 Excellent

✓ STM32 hardware demonstrates real-time determinism suitable for safety-critical control loops

⚠️ Linux kernel introduces scheduling variance; outliers caused by OS, not CAN hardware

CAN BUS LATENCY ANALYSIS

Key Findings & Recommendations

System Assessment

- Motor Control Loops: 789 μ s RTT suitable for 10–100 Hz control
- Safety-Critical: STM32 determinism excellent for ASIL-compliant systems
- Hardware Performance: CAN transceiver + CAN solid at 500 kbps



**Proposed
Upgrade:**

1000 kbps

SPIKE: NATIVE AGL KUKSA & C++ INTEGRATION

Migration from direct CAN to VSS-based Databroker architecture

WHY KUKSA?



CAN YOU AT LEAST
TELL ME WHY?

SPIKE: NATIVE AGL KUKSA & C++ INTEGRATION

perplexity.ai

Critério	KUKSA	SOME/IP	FastDDS	MQTT	Zenoh
VSS Standardization	✓ SIM	✗ NÃO	✗ NÃO	✗ NÃO	✗ NÃO
Avg Latency (ms)	< 1 (teórico, gRPC)	7–8 (Navet 2015)	5–6 (ECRTS 2023)	100–300 (Azure)	? (não verificado)
Max Latency (ms)	< 1 (teórico)	?	17 (WATERS 2019)	1000+	?
Jitter	?	?	±1–2 ms (6–14% relativo)	Alto (>50%)	?
AGL Native Support	Excelente	Bom	Emergente	Bom	Emergente
AUTOSAR Compliance	Moderado	Excelente	Nenhuma	Nenhuma	Nenhuma
Determinismo (ASIL)	Moderado (AGL)	Bom	Bom	✗ Não	?
Resource Overhead	~100 MB	~50 MB	~150–200 MB	~20 MB	?
Cloud Ready	SIM	SIM	SIM	SIM	?

SPIKE: NATIVE AGL KUKSA & C++ INTEGRATION

- KUKSA Latência Teórica (gRPC) – gRPC Specification (HTTP/2 based) ;
- SOME/IP Latência & Resource Overhead – Nicolas Navet, “Time-Triggered vs Event-Triggered Real-Time Systems”, SAE Preprint 2015 (benchmark automotive com vSomeIP, 7–8 ms, ~50 MB) ;
- FastDDS Latência & Jitter – Sciangula et al., “Real-Time DDS Middleware Evaluation for Automotive Systems”, ECRTS 2023, ETH Zurich (Dell Optiplex 7070, Ubuntu 20.04, 3 topics, 1 kB, 50k samples, 5.4 ms, jitter 6–14%, max 17 ms com WATERS 2019 Bosch) ;
- MQTT Latência & Resource Overhead – Microsoft Azure IoT Operations Official Benchmarks (100–300 ms média, 1000+ ms máx, jitter >50%, ~20 MB) ;
- KUKSA AGL Native Support – COVESA Official Documentation & AGL Integration (Docker pronto, integração VSS nativa) ;
- SOME/IP AGL Support – AGL Documentation & AUTOSAR Stack (suportado, requer stack AUTOSAR) ;
- FastDDS AGL Support – AGL + OpenDDS communities (experimental em AGL, foco em Ubuntu/Fedora) ;
- MQTT AGL Support – Mosquitto MQTT Broker (open-source em AGL Linux, repositórios padrão) ;
- Zenoh AGL Support – Eclipse uProtocol Integration / Zenoh project roadmap (estado emergente, integração em desenvolvimento) ;
- SOME/IP AUTOSAR Compliance – AUTOSAR Standard Specification (Classic & Adaptive), journals.plos.org (middleware nativo)
- AUTOSAR, usado em ECUs críticos) ;
- KUKSA Determinismo ASIL – gRPC over Linux + AGL PREEMPT_RT patches (moderado sem RT, pode atingir ASIL-C/D com PREEMPT_RT) ;
- SOME/IP Determinismo ASIL – AUTOSAR Safety Standards + vSomeIP benchmarks (excelente com Ethernet TSN, jitter 5–10% até ASIL-C, possível ASIL-D) ;
- KUKSA Resource Overhead – gRPC specification + GitHub KUKSA (estimado ~100 MB, não medido) ;
- FastDDS Resource Overhead – ECRTS 2023 + multi-threading analysis (estimado ~150–200 MB, não especificado no paper) ;
- KUKSA Cloud Ready – KUKSA.cloud & gRPC/HTTP2 architecture (cloud-first, suporta Google Cloud, AWS, Azure, integração VISS) .

SPIKE: NATIVE AGL KUKSA & C++ INTEGRATION

CAN vs KUKSA Databroker Performance

```
./latency_analyzer latency_sent_kuksa.txt latency_received_kuksa_1000.txt
Method: Nearest Time Match (Steady Clock Precision)...
[Sender] Loaded: 1000 packets.

== PERFORMANCE REPORT ==
Sender Lines: 1000
Receiver Parsed: 1001
Receiver Matched: 1000

-----
Packet Loss: 0.00 %
Avg Latency: 2.212 ms
Min Latency: 0.825 ms
Max Latency: 37.023 ms
```

```
./latency_analyzer latency_sent_kuksa.txt latency_received_can_1000.txt
Method: Nearest Time Match (Steady Clock Precision)...
[Sender] Loaded: 1000 packets.

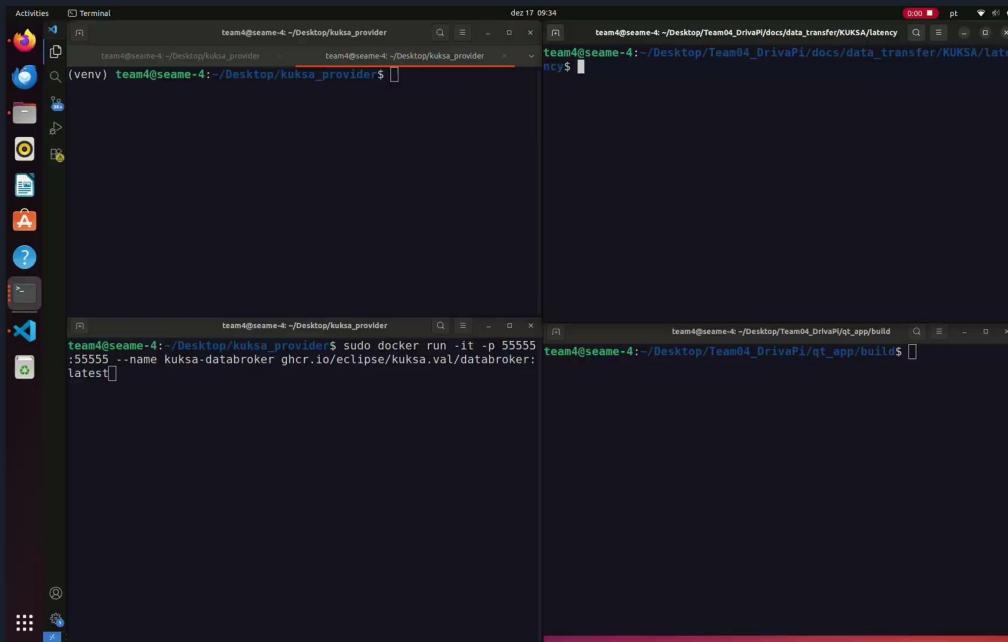
== PERFORMANCE REPORT ==
Sender Lines: 1000
Receiver Parsed: 1000
Receiver Matched: 1000

-----
Packet Loss: 0.00 %
Avg Latency: 0.148 ms
Min Latency: 0.033 ms
Max Latency: 0.763 ms
```



SPIKE: NATIVE AGL KUKSA & C++ INTEGRATION

CAN vs KUKSA Databroker Performance



The screenshot shows a terminal window with three tabs open, running on a Linux desktop environment. The desktop interface includes a dock with icons for a browser, file manager, terminal, and other applications.

- Tab 1:** team4@seame-4:~/Desktop/kuksa_provider\$
- Tab 2:** team4@seame-4:~/Desktop/kuksa_provider\$
- Tab 3:** team4@seame-4:~/Desktop/Team04_DrivaPi/docs/data_transfer/KUKSA/latency\$

In the bottom-left tab, the user runs a Docker command to start a KUKSA databroker:

```
team4@seame-4:~/Desktop/kuksa_provider$ sudo docker run -it -p 55555:55555 --name kuksa-databroker ghcr.io/eclipse/kuksa.val/databroker:latest
```

In the bottom-right tab, the user is navigating through a directory structure related to a Qt application build:

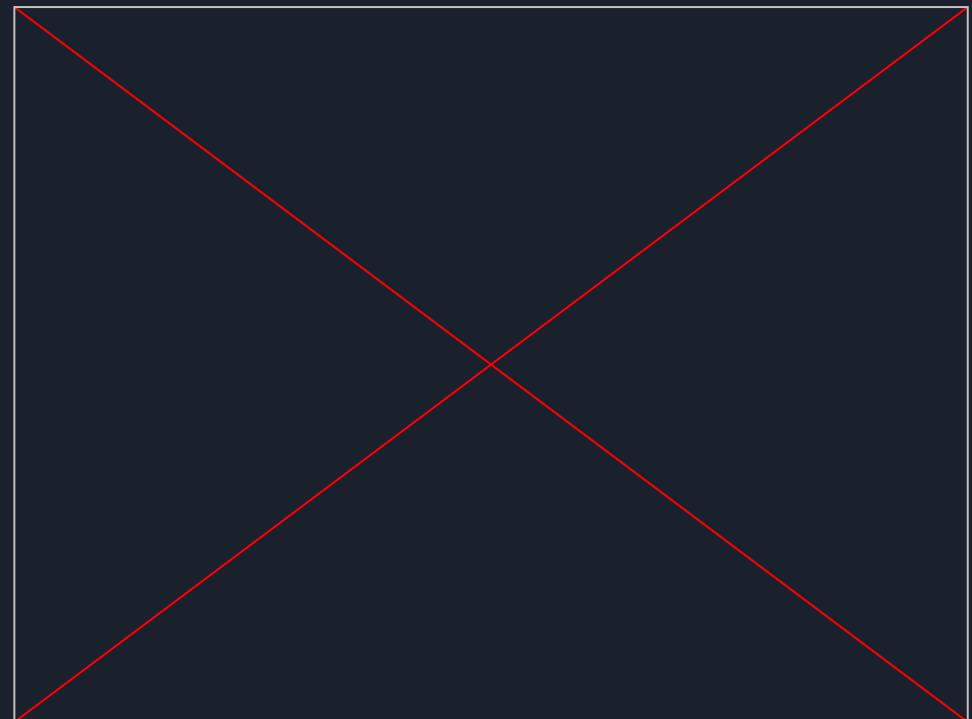
```
team4@seame-4:~/Desktop/Team04_DrivaPi/qt_app/build$
```

Raspberry Pi/AGL CAN Communication

Configure and validate the **CAN interface on a Raspberry Pi running AGL**. This includes enabling **SocketCAN**, bringing up the **CAN interface with the correct bitrate**, and verifying **bidirectional communication with the STM32 using command-line tools**. A minimal C/Rust application is used to confirm application-level access to the CAN interface. **This ensures the AGL application layer can communicate with the STM32 control board.**

Joystick Control DC MOTOR

Rust-based CAN communication and a joystick-controlled DC motor controller, enabling real-time motor control via **CAN ID 44** using a **ShanWan gamepad**. It includes a simple CAN message sender and a full controller application that maps joystick input to motor direction and speed.





RUST CROSS-COMPILEMENT FOR AGL

Set up the **host development environment** to cross-compile Rust applications for AGL running on a Raspberry Pi. This includes **installing the required Rust target** and **cross-compilation toolchain**, building a minimal Rust proof-of-concept application, and validating that the **generated binary runs correctly on the AGL device** without linker or dependency errors. **This ensures Rust applications can be safely and reliably developed for the AGL platform.**



In sum...

Thank you!

