Team 6 (Personal Data Aquisition

Aidan Agee

Blake Babb

Jake Goodwin

Patrick Iacob

Project Partner: Chris Patton

Contents

Abstract

The aim of this senior capstone project is to design and implement a versatile data logging system that integrates single board computers (SBCs), STM32 microcontrollers, and the Rust programming language. The system will be capable of collecting, storing, and processing data from various sensors and input sources, providing a flexible and reliable solution for a wide range of applications. The project will involve the development of software for both the SBCs and STM32 microcontrollers, leveraging the unique capabilities of each platform. The Rust programming language will be used for its safety, performance, and ease of use, allowing for efficient and reliable code development.

1. Selection and integration of sensors and input devices for data collection.
2. Design and implementation of communication protocols between SBCs and STM32 microcontrollers.
3. Development of data processing algorithms and storage mechanisms.
4. Creation of a user interface for system configuration and data visualization.

The project will culminate in the deployment and testing of the data logging system in real-world scenarios, demonstrating its effectiveness and reliability. The system will be designed with scalability and modularity in mind, allowing for future expansion and customization. Overall, this project aims to provide a comprehensive solution for data logging applications, showcasing the capabilities of modern embedded systems and the Rust programming language in real-world applications.

Change Log

.. okay

Product Requirements Document

Authors: Jake Goodwin, Aidan Agee, Blake Babb, Patrick Iacob

DATE: 2023-11-23

Problem Description

Existing personal data acquisition devices are either too expensive or too DIY for most potential users. Many are designed for aerospace, automotive or research purposes, and are too expensive and unnecessarily complicated for casual users. The only other option is for users to build their own devices from prefabricated parts, which is too complicated and requires too much much prerequisite knowledge for most potential users.

Scope

The scope of this project is to develop a prototype for a personal data acquisition device. This prototype will have the ability to collect real time data from a variety of sensors, including accelerometers, gyroscopes, GPS modules, and thermometers. These components will need to combined in a printed circuit board for the final prototype. The scope of this project also includes development of a web-based UI that both presents the data gathered by the prototype and sends commands to the physical device to record data and configure sensors.

Use Cases

The user will take the product along with them on an outdoor activity and subject it to normal conditions for that activity.

The user will connect the product to a phone or laptop they brought with them, and view the data stream and take samples using the user interface, and save the data locally.

The user will choose and connect selected modules to the system using the CAN(controller area network) bus.

Purpose and Vision(Background)

Our purpose is to develop a personal data acquisition system that records all the data a user might want, and is cheap and easy to set up and use. It should be able to record data on acceleration, force, position, etc. require minimal setup, and can be hooked up to bike, go-kart, etc.

Stakeholders

Capstone Team

The capstone team are the main decision makers for the project, and will need extensive information for the product's requirements and implementation details. They will also need oversight from the project partner and TA.

Project Partner

The project partner will be working very closely with the capstone team, and will need to know the teams capabilities and status, and the status of the project.

Project TA

The TA needs to be informed on project progress and any issues the team may be having.

Capstone Instructors

The instructors require much of the same information as the TA, but because they are working less closely with the team there is less urgency.

Users

Users will need to know the product's capabilities, limitations and intended use.

Preliminary Context

Assumptions

- We have a suitable power supply of 12v to power the system.
- The end user has a device capable of connecting to an ad-hoc network.
- The data to be logged doesn't require more speed than the CAN 2.0 standard.
- The environment it's meant to be used in is electrically noisy.

Constraints

- As undergraduate students, our team has limited experience in the field, so we will have to learn a lot to deliver the product.
- Our budget is limited, so we will have to choose components carefully based on price.
- We are limited to three terms to deliver our product.

With these constraints factored, the biggest concerns for the feasibility of our project are the skills that need to be learned and limited time alloted to do so and complete the project. As an example, the team has primarily non-formal experience in hardware organization but has thus far worked efficiently in that aspect of the project. These risks are mitigated by the expertise and technical support offered by our project partner, and we consequently find the scope of our project realistic.

Dependencies

- The rust language, (reduces bugs and helps with memory safety.)
- C compiler(s), (C ABI is still used as a way to interface with libs.)

- Rust Embassy Library. (Embedded rust lib to reduce boilerplate)
- Rust Rocket(web server)
- STM SDK and HAL (Good refences for the actual hardware.)
- The CAN standard.
- The Unix networking stack
- SQLite and or rust file I/O
- Rust Libraries available for individual sensor modules.

Some possible bottlenecks that could occur given our current dependencies would be centered around sensor modules not having an existing library written in rust. This would add more development time to the project. However, we've researched workarounds and discovered tools to generate the needed interfaces for rust from a C header file.

Market Assessment and Competition Analysis

RexGen: Proprietary CAN bus based data logger, hard to find tutorial or documentation and is prohibitively expensive for hobbyists. Also unable to guarantee that their system is memory safe.

CANedge1: CANedge1: It has open source elements to it and documentation that is accessible, but still does not meet the requirements for its cost. DEWEsoft sells test and measurement equipment. Their products are not a good fit for our users because they are designed for industry, and therefore overkill and are prohibitively expensive for an individual. Omega Engineering sells data loggers that can record the data our users would want, can connect to a remote device over Bluetooth and have easy to use interfaces. However most of their data loggers only record one or two types of data, so a user would need to buy many of them, which would be inconvenient and expensive. An Apple Watch can track a user's activity data, and send it to an iPhone with an easy to read interface. However, the Apple Watch is limited in what kind of data it can record, and would not be appropriate for our users due to its many other unneeded functions. There are guides on the internet that instruct a user on how to build their own data acquisition device using Arduino or Raspberry Pi microcontroller much more inexpensively than the other alternatives. However, this requires the user to have background knowledge in circuitry and programming, and requires a lot of time and effort to set up.

Target Demographics (User Persona)

Terry is an amateur Go-kart enthusiast who was brought into the hobby 8 months ago by friends and has become entrenched in the hobby since then. They are looking for a way to improve their performance but need more information about their current racing habits to do that. Alice is a CTO of a large company that has decided to data log the forces and location their products experience during shipping through multiple contracted pilots and routes. She needs a system that isn't cost prohibitive to deploy in large numbers and can be customized for her company's other projects as needed. John is an extreme snowboarder looking to collect data from his downhill tricks in order to help his friend create realistic and smooth animations for a snowboarding video game. He needs a data logging system that can endure cold environments and is modular so he can keep down the bulk/weight of the system while carving toeside and hitting some sweet jumps. James is a competition mountain biker who wants to record and analyze data during rides for performance improvement. Uses a smartphone and needs an easy-to-use interface. He needs a system to compare data between runs.

Requirements

User Stories and Features (Functional Requirements)

User Story	Feature	Priority	GitHub Issue	Dependencies
As a mountain biker, I want to be able to view my instantaneous speed at any point in my journey.	Gps	Must Have	TBD	Common firmware.
As a motorsport hobbyist, I want to be able to record the g-forces I experience while going around tight corners.	Accelerometer.	Must Have	TBD	Common firmware.
As a winter sports enthusiast, I want to be able to track the turning speed of my snowboard.	Yaw rate sensor.	Must Have	TBD	Common firmware.

Non-Functional Requirements

- Delay on data transmission should be at an acceptable level
- Code should be well documented, following coding standards and best practices
- User interface should intuitive and fast to use
- The product should use security best practices whenever possible

Data requirements

- Analog data will be converted to digital.
- Sensor data must be reliable, resistant to EMI
- Sensor modules must adhere to the CAN protocol.

Integration requirements

- All interfaces will be rust doc documented.
- Tests will ensure API usage integrity.
- All modules that are to use the interface must pass integration tests.

User interaction and design

- Use a web server to interface with the user
- Can display live data, configure sensors, and download data
- Build using EGUI and Rust Rocket
- Should focus on ease of use for less experienced users
- Should be able to clearly provide all data and provide configurability

UI Mockup from project partner

User Documentation

The user documentation will be produced in markdown or LaTeX into a PDF or webpage. This documentation will cover the basic usage of the system and instructions on how to build it.

Testing and Quality Assurance

Testing will be done through TDD(test driven development) using the supported testing frameworks for rust and C. These tests will allow us as developers to ensure the assumptions we make about our code matches the actual behavior of it.

Quality assurance will mostly be handled by adhearance to style standards enforced by the lanuages LSP(language server protocol) servers. The two that will see extensive use in this project being:

1. Rust-analyzer
2. clangd

Bug and issue tracking will all be handled by GitHub's Issue and project system. This also serves as a way to allow public contributions in the future to the code base.

Milestones and Timeline

Item	Description	Duration
Schematics	The wiring schematics	2 months
PCB	PCB gerber files	1 month
uC	Firmware for STM	
Sensor FW	Sensor module firmware	
UI	Web user interface	
Server	The back end web server	

Open questions
Currently, there are no open questions in the project.

Out of scope
- Support for more than the listed sensors.
- HAL development
- Radiation Hardening
- Full EMI sheilding
- Full support for 10 channels saturated with sensor data at 5kHz
- Water resistance at any depth or submersion
- Documentation beyond rustdocs/doxygen and markdown/latex.
- Wireless connectivity beyond ad-hoc wifi.

Software Design and Architecture Document (SDA)
Authors: Jake Goodwin, Aidan Agee, Blake Babb, Patrick Iacob

DATE: 2023-12-03

Introduction
This document establishes the software and hardware architecture for a personal data acquisition system. Selecting the correct architecture improves development velocity and enables more robust functionality by having systems that support each other. Cohesive Software Architecture is particularly important for this project as our workflow has contributors split into two sub teams which are developing a frontend and backend that must communicate. In addition to that, designing an appropriate hardware architecture will reduce development costs and help our product find its place in the market.

Architectural Goals and Principles
Our hardware and software architectures have the primary goal of being synergistic with each other and possessing libraries that interface with each other.

Because our project covers layers ranging from hardware to a website, connection throughout the stack is critical. Additionally, as we work through the prototyping stage our architecture will practice modularity to be able to add new sensor hardware as we expand the capabilities of the product. Our architecture does not need to prioritize scalability as the final objective is only to develop a prototype.

System Overview

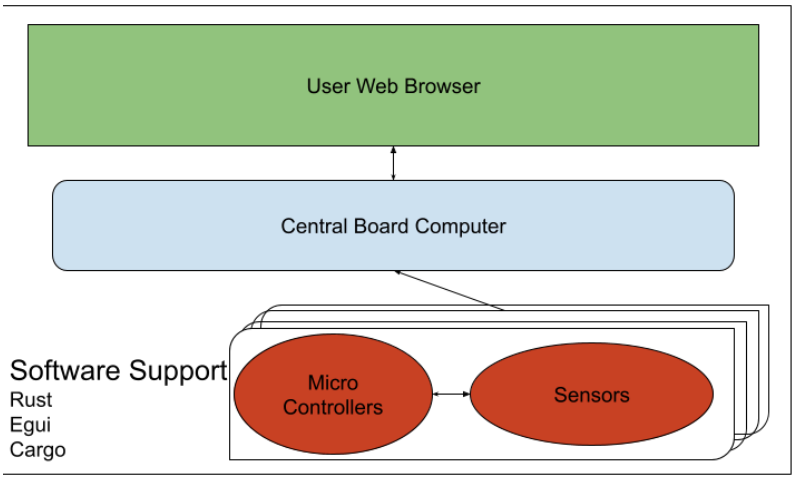

Figure 1: image

Architectural Patterns

Controller Responder: This pattern is useful as our hardware SBC can act as the controller which has a single responder in the user webpage. This pattern would help cache data generated by the controller to provide a seamless experience to the user in the face of latency issues. Additionally, the ability to access data in the responder without affecting the controller will allow for computation in an environment separate from the hardware board.

Event Sourcing: This pattern works especially well with real-time data, which is what this project is all about. Our hardware controller can act as the producer and broadcast its data to a web server which will act as the event source for an y users that want to query the server and consume that data. The fail-safety of this design is also important to this project as data acquisition environments, such as racing or aeronautics, often cause damage to the controller while still requiring data to be accessible.

Component Descriptions

Sensors: Hardware components that acquire the raw data, such as accelerometers o r GPS devices.

Microcontroller: Small computer that is responsible for coordinating the sensors and collecting their data to be broadcast to the web server.

Web server: Acts as the intermediary between the user and the physical acquisition device. Communicates with the board, composed of the microcontroller and its sensors, to collect data which it then relays to the user interface when queried.

User interface: An HTTP webpage that requests data from the web server to present in useful ways to the user.

Data Management

Sensor readings are transmitted over canbus in JSON format. Data is stored in a relational database on the Raspberry Pi. RESTful API endpoints are provided for CRUD operations on data.

Interface Definitions

There will be a user interface to collect data from each sensor and display it to the user. There will be interactions to get event logs from each sensor, and to clear the event logs. The user interface will be hosted on a web server, which users will connect to over with their browser over HTTP.

API endpoints for the web interface include:

- `GET /data`: Returns a list of collected data from personal devices.
- `GET /sensors`: Returns a list of sensor configurations.
- `POST /data`: Allows the addition of new data.
- `PUT /data/{id}`: Updates data with a specified ID.
- `PUT /sensors/{id}`: Configure a sensor with a specific ID.
- `DELETE /data/{id}`: Deletes data with a specified ID.

Considerations

Security

The primary data security risk in this project is data loss due to physical conditions of the board. This includes both permanent damage through the elements or impacts as well as location preventing broadcast to the web server. A caching system on both the board and in the web server is the approach that will be used to mitigate this risk.

The data security risks due to bad actors in this project are minimal as the data being processed is kinematic information. Regardless, our web server will require password authentication to access the RSA encrypted data.

Performance

There are two primary performance concerns of the product. The first is the resolution of our data and how quickly we can poll our sensors, for which the current target is acquiring 10 data points per second. We plan to achieve this metric by screening hardware before they are implemented into the design to ensure it can meet this desired performance. The second concern is with the stability of the connection between the user interface and the board's raw data. We plan to create a web server that will be able to cache the data produced by the board and present to the user at will to mitigate this concern.

Maintenance and Support

Once the prototype is complete, maintenance and development will be inherited by Patton Dynamics, the company partner for this project. The company has a background in aeronautics, competitive motor racing, and computer assisted physics, all of which are relevant to the project area. Their experience with the common end users of personal data acquisition devices makes them very capable of supporting users through the life cycle of the product.

Deployment Strategy

As the ultimate objective for this project is to develop a prototype PCB that hosts a local webserver as a user interface, there is only deployment in a development environment.

Testing Strategy

Software (SBC side) Testing

User testing will be done to ensure users can understand and use the interface effectively. These tests should be focused on confirming that functional requirements are met.

Integration testing will be done with mock data until microcontrollers and sensors are operational. Further tests will be conducted when hardware is more complete.

Firmware Testing
Our firmware testing methodology will make heavy usage of mocks for many of the hardware components so we tests can be run on development machines instead of on the embedded systems.

A Red Green refactoring/testing cycle will ensure we always know the tests we write are both useful and logically possible to fail. Writing any tests that cannot fail would end up being dead or uncalled code.

Many of the usual tests that would be prevalent for ensuring good memory management will be unnecessary from our use of the rust language. This along with the built in `rust-docs` will allow us to even use our tests as examples where needed as part of our documentation.

Integration testing will mostly be handled as mocked interfaces replicating the physical hardware that will be required to collect the data. Further tests can added as needed should more sensors be added to the project at a later point in time.

Glossary
- SBC: Single Board Computer
- Rust: A modern compiled and memory safe language
- PCB: Printed Circuit Board

Software Development Process(SDP)
Authors: Jake Goodwin, Aidan Agee, Blake Babb, Patrick Iacob

DATE: 2023-11-16

Principles
- We will respond to asynchronous communication within 24 hours
- We will be at meetings on time and pay attention
- All changes need to be isolated to their own git branch
- Each work item need a corresponding GitHub issue
- Pull Requests have to be reviewed by at least one team member
- We will use a kanban board to continuously work on the backlog
- Once a work item is complete, a pull request is created
- Blocks need to be discussed as soon as possible

Process
Task Selection: * Kanban style * Select highest priority item within your role's domain

To Solve an Issue and Meet Acceptance Criteria: 1. Write failing tests.
2. Write code to pass tests. 3. Repeat. 4. Open a pull-request and merge

Steps in software development
1. Create a github issue/milestone.
2. Assign github task/issue.
3. Create fork of repo.
4. Write tests using the test framework.
5. Push the tests to the fork (optional).
6. Write Code to pass the tests.

7. Test the code.
8. Push to the fork repo.
9. Create a pull request with description.
10. Get approval for the PR(pull request).

Goals & Objectives

Develop a personal data acquisition system that records all the data a user might want, and is cheap and easy to set up and use. * Record data on acceleration, force, position, etc. * Minimal setup * Can be hooked up to bike, go-kart, etc.

Project Scope

- Design of simple UI to display data.
- Design of hardware/schematics for system.
- Firmware for sensor modules in rust.
- SBC with rust software to store/log sensor data over CAN.

Roles

ROLE	PERSON	RESPONSIBILITIES
UI	Blake	Develops the web page front end
SBC/SW	Aidian	Develop logic to relay sensor data to UI
FIRMWARE	Patrick	Develop firmware for microcontrollers
HARDWARE	Jake	Design schematics, wiring diagrams & PCB files

These are the general outlines for the four different roles in the project. We have a verbal agreement at the moment that we will help out with parts of the project outside our roles as needed.

Tooling

Purpose	Name
Version Control	Git
Project Management	GitHub Projects
Documentation	Rustdocs & MD
Test framework	Rust & Cmocka
Editor	ANY
Schematics & PCB	KiCAD
Communication	Discord/Teams/Email

Version Control

Git will allow our team to track changes in the projects files over time. Also prevents the loss of work from hardware failures.

Project Management

GitHub projects is integrated into github organizations as well as git. The project management software makes the collaboration between developers easy and will make tracking milestones and issues for the entire project across multiple repositories a possibility.

Documentation

Documentation will primaily be done through the built-in rust-docs feature. This is accesiable via the CLI(command line interface) tooling. This will encapsulate how the code itself and any interfaces are documented.

Because the documentation is genreated as part of the code this will ensure that up to date and accurate documentatio is always availble.

Secondary documentation meant for non-developers will be done using a combination of markdown and LaTex where needed. This will be availble usally in a PDF format.

Schematics & PCB

The KiCAD program gives access to the schematics and PCB designs to all team members due to the software begin free and open-source.

It will allow us to comment, label and design the needed circuits for the physical hardware of the system; providing a good troubleshooting resource as well.

Communication

Discord: * Used to coordinate team meetings. * To share ideas/brainstorm * give updates on project.

Teams: * TA meetings.

GITHUB: * To discuss project issues. * share documentation.

Definition of Done(DOD)

- Acceptance criteria all satisfied by code changes
- Changes have been merged to master after completing the Pull Request Process
- A completed Pull Request has at least one approval and no marks for “Needs Work”
- All tests pass with changes implemented and no reversion is required
- Relevant documentation for the feature has been updated
- Discussion points are prepared for next meeting

TESTING

Rust:

The testing for all code repositories will be done using a testing harness or framework. For rust this takes the form of the `cargo test` command, which is part of the package managment system(tool-chain).

These tests will be used as one of acceptance critera for a branch to be pulled into the main branch.

C:

Some libraries or areas where the use of C code is needed we plan to use cmocka as the unit testing framework. This combined with Cmake as the build system will give us a host agnostic development cycle.

Quality Assurance

Quality assurance will mostly be handled by adhearance to style standards enforced by the lanuages LSP(language server protocol) servers. The two that will see extensive use in this project being:

1. Rust-analyzer
2. clangd

Feedback

Feedback on the work done will take place in the github projects. The issues and discussion boards are the main locations for this, with the weekly meetings and discord being a secondary and informal medium for minor feedback.

Release Cycle

For the moment we will used semantic versioning with the standard Major.minor.patch format. This will help when it comes to dealing with any major changes that break APIs.

Contingency Plans

Feedback can be shared during weekly standup with the TA, or over Discord if they are more time-sensitive, after which it should be reviewed by the whole team, and then incorporated. Changes to the whole process will require more comprehensive feedback and approval from the team before going into effect, after which related documents should be modified as soon as possible.

In the event of unexpected challenges, the team should be notified immediately, and if serious enough should be brought up with the TA, project partner or instructor, otherwise they should be brought up during regular meetings.

Timeline

- 12/15/2023: Version 0 complete with breadboard organized hardware and visual UI elements
- 03/22/2023: Version 1 complete with functionality between firmware, SBC, and UI

Environments

Environment	Infrastructure	Deployment	What is it for?	Monitoring
Production	Github releases	Release	Packaging install files.	N/A
Staging	Github actions			Github Pull requests
Development	Local	Github commits	Development and unit tests of microcontroller-based sensors	Manual