|  |  |
| --- | --- |
|  | Proprietary  Information |

Current sensor graphing application

Software Design Specification Template

12th June, 2023

Version 0.0

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Date | Description of Changes | Author |
| 0.0 | 12th June, 2023 | Created | Dhruv |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Introduction

This document provides a detailed description of the design and architectural decisions for a Python-PyQt application that interfaces with an Arduino device for data acquisition, visualization, and storage. It offers an overview of the system's functionality, the system's static and dynamic views, and design strategies.

System overview

The system is composed of an Arduino device connected to a computer running a Python-based GUI application. The Arduino collects environmental data through its sensors and transmits it to the Python application over a serial connection. The Python application processes this data, visualizes it in real-time, and provides controls for data management.

A summary of the requirements

Functional Requirements

* System Inputs:
  + User interactions with the GUI; Arduino data received over a serial connection.
* System Outputs:
  + Real-time visualization of the data; data files when the user chooses to save the data.
* System Processing Needs:
  + Real-time processing of data from the Arduino; GUI updates based on user interactions.

Non-functional requirements

* Response Time:
  + The application should operate with minimal latency to ensure real-time data visualization.
* Expected Accuracy:
  + The application should accurately display the data received from the Arduino device.
* Memory Footprints:
  + The application should have a minimal memory footprint to ensure it can run on a variety of computers.

Strategies (according to webapp)

Design Strategy

* Development Approach: We will adopt an iterative and incremental development approach, similar to Agile methodologies. This approach allows us to be flexible, receive feedback early on, and continuously improve the application. We will develop features in iterations, ensuring that they are tested and integrated incrementally.
* Adoption of Technology: We have decided to use QT5 due to its interactive functionality on the graph which is useful for zooming, panning and intuitively easier.
* Object-Oriented Approach: Our design follows an object-oriented approach, which promotes modularity, reusability, and maintainability. We are also using the function oriented approach for dealing with Arduino’s device data. We encapsulate related functionalities into classes, ensuring that each class has a specific responsibility and is easy to understand and modify.
* SDLC: We will follow an Agile software development methodology, which allows us to iterate quickly, collaborate closely with stakeholders, and adapt to changing requirements. This approach emphasizes continuous integration, frequent communication, and iterative testing, enabling us to deliver working software in shorter cycles.
* Prioritization: We will prioritize development, integration, validation, and delivery milestones based on the critical functionalities and dependencies. We will focus on core features such as graphing, zooming, saving data and image and more features as described.
* Choice of Programming Language: We have chosen Python as the programming language for the standalone application. It provides us with the flexibility to integrate with different services and technologies. For Arduino code, we have used C to interact and upload the sketch to the circuit.

These design strategies aim to ensure a scalable, modular, and maintainable web application that meets the functional and non-functional requirements. We aim to leverage appropriate technologies and methodologies to deliver a high-quality user experience.

Assumptions/Constraints

#### Business assumptions

* The project will be completed by a single developer.
* The project is expected to be completed within a reasonable timeframe.
* The budget/cost for the project is not provided.
* The scope of the project is defined in the SRS document.

#### Technical assumptions

* The application is expected to perform accurately and reliably in a variety of operating environments.
* The application will be developed using Python and PyQt.

Dependencies

The critical dependencies for the system are as follows:

1. Operating System: The standalone and crossplatform should be designed to run on multiple platforms, such as Windows, Linux and.
2. Server: Hosting is not required.
3. Framework: QT5 allows frontend interaction, combined with Pythons functionalities to detect these interactions and act on it.
4. Libraries: QT5, serial, time, numpy, sys, json, os and more
5. Sub-systems: The system interacts with external sub-systems such as the Current sensor board. It relies on proper communication through a com port and integration with these sub-systems to facilitate data reading and graphing capabilities.
6. Platform: The application will not be dependent on other systems except a circuit board to run. It should be able to run on windows, linux and MacOS.

Test strategies

The following test strategies can be adopted to ensure the quality and reliability of the system:

1. Unit Testing: Conduct unit tests to validate the functionality of individual components, functions, and methods within the system. This helps identify and fix any bugs or issues at a granular level.
2. Integration Testing: Perform integration tests to verify the seamless integration and communication between different system components, including the application with the Arduino circuit and other functionalities like disconnect with start and so on. This ensures that the components work together harmoniously.
3. Functional Testing: Validate the system against the functional requirements to ensure that it performs as intended. Test each feature and functionality of the application.
4. User Acceptance Testing: Involve end-users or stakeholders in testing the system to ensure it meets their expectations and requirements. Gather feedback and make necessary improvements based on user feedback to enhance the user experience.
5. Performance Testing: Evaluate the system's performance under expected loads and stress conditions. Measure response times, resource utilization, and system stability to ensure it can handle the expected user traffic and deliver a smooth user experience.
6. Security Testing: Perform security testing to identify and address potential vulnerabilities in the system. This includes testing for data protection, secure communication, and user authentication to ensure the system safeguards user data and maintains secure operations.

****System Deployment Environment****

The deployment plan details specific to your project are as follows:

1. Environment Details: Development Environment: The system will be developed in an environment where python 3.0 and PyQT5 are installed. The production environment will run a executable of the application. The executable will be packaged using pyInstaller on the .py code which will package all libraries and dependencies in a single .exe. The user's computer should have a USB port to connect the Arduino device.
2. Schedule (Milestones): The key milestones in the development plan may include:
   * Completion of requirements gathering and analysis
   * Design and architecture definition
   * Implementation of core functionalities, graphing capabilities, save and load features, zoom and pan capabilities and having open ended capability to see the historical data.
   * Integration of Arduino device to the host device
   * Testing and quality assurance, including functional testing, usability testing, and accessibility testing
   * User acceptance testing and feedback incorporation
   * Deployment to the production as an executable.
3. Rolling Plan Process: The rolling plan process for the project may involve the following steps:
   * Conducting an initial pilot release to gather early feedback and validate the system's usability and accessibility.
   * Incorporating feedback and making necessary improvements to enhance the user experience and address any identified issues or bugs.

Rollback Process: In the event of critical issues or unexpected problems during a release, a rollback process will be initiated. The rollback process will include:

* Reverting to a previously stable version or state of the system that was known to be functioning properly.

Conducting thorough testing and investigation to diagnose and resolve the issue before proceeding with another release.****High level design (HLD)****

****Static View (structural details)****

****System decomposition****

In order to ensure a solid system structure, we have devised a plan to break down the system into smaller, manageable components. Here's our approach:

1. System Decomposition: We will divide the system into several subsystems to handle specific tasks. These subsystems include:
   1. **Arduino Module**: This module is responsible for gathering data from the Arduino device and sending it to the Python application.
   2. **Data Processing Module**: This module processes the incoming data from the Arduino device. It handles data conversion, error checking, and data preparation for visualization.
   3. **Data Visualization Module**: This module is responsible for plotting the data in real-time and updating the plots based on user interactions.
   4. **User Interaction Module**: This module handles all user interactions with the GUI.
2. Functional vs Object-Driven Decomposition: Our focus will be on the functionalities each subsystem provides. However, we'll also consider the entities involved, such as the Arduino module, the data processing module, and others.
3. High Modularity and Abstractions: We'll aim for a highly modular and abstract design by breaking down the functionalities into separate modules within each subsystem. This will make our code more organized, reusable, and easier to maintain.
4. Low Coupling: We'll strive to keep the modules within each subsystem loosely coupled to minimize dependencies and enhance flexibility. The systems will be highly coupled due to being a real time graphing application which requires constant reading from the Arduino device.
5. High Cohesion: Each module within a subsystem will have a high degree of cohesion, meaning that its internal components and operations will work closely together to achieve specific functionalities. This will improve code readability, maintainability, and overall understanding of the system.
6. Reusability: We'll identify specific modules or components within the system that have potential for reuse, both within the same system and in future projects. For example, the read data function has to be reused every time data comes in to parse data from the circuit through the USB cable.

****System Architecture****

**The system follows a Client-Server architecture, with the Python application acting as the client and the Arduino device as the server. The Python application sends commands to the Arduino device, which responds with the requested data.**

Data model, data design and data structures

The application uses in-memory data structures to store the data received from the Arduino device. The data is stored in a time-series format, with each data point consisting of a timestamp and the corresponding measurements.

Class Diagram

The application is developed using Python's object-oriented features. The main classes in the system are **AppWindow**, **ArduinoData**, **GraphWindow**, and **ArduinoDevice**. A more detailed class diagram will be provided in the Low-Level Design (LLD) section.

## Dynamic View (Behavioral Details)

#### **5.2.1 Modes/States**

The application has two main states: "Idle" and "Data Collection". In the "Idle" state, the application is waiting for user interaction. In the "Data Collection" state, the application is actively collecting and visualizing data from the Arduino device.

#### **5.2.2 Use Cases**

Refer to the Use Case section in the SRS document for detailed use cases.

#### **5.2.3 Data Flow**

Data flows from the Arduino device to the Python application over a serial connection. The Python application processes and visualizes this data.

#### **5.2.4 Sequencing**

Data collection starts when the user interacts with the GUI to start data collection. The Arduino device starts sending data, which is processed and visualized in real-time by the Python application.

#### **5.2.5 Design Risk**

A potential risk is the real-time processing and visualization of data, which could be impacted by high data rates or resource limitations on the computer. Mitigation strategies include efficient data processing algorithms and providing the user with control over the data rate.

#### **5.2.6 Error Handling**

The application will include error checking and exception handling mechanisms to prevent and handle potential errors. Errors will be logged and displayed to the user through the GUI.

1. ****Low level design (LLD)****

**This section provides a detailed description of the low-level design and internal workings of the Python-PyQt application.**

* 1. **APIs and Return Values**

**AppWindow: This is the main GUI window. It contains the following functions:**

**\_\_init\_\_(self): Initializes the GUI and sets up the initial state.**

**start\_data\_collection(self): Starts the data collection process by sending a command to the Arduino device.**

**stop\_data\_collection(self): Stops the data collection process.**

**save\_data(self): Saves the current data to a file.**

**load\_data(self): Loads data from a file and displays it on the plot.**

**update\_plot(self, data): Updates the plot with the new data.**

**refresh\_ports(self): Refreshes the list of available COM ports.**

**connect\_device(self): Establishes a connection with the Arduino device.**

**disconnect\_device(self): Disconnects from the Arduino device.**

**Worker: This class manages the thread that reads data from the Arduino device. It contains the following functions:**

**run(self): Reads data from the Arduino device in a loop until stopped.**

**stop(self): Stops the data collection loop.**

**GraphWindow: This class handles the data plotting. It contains the following functions:**

**\_\_init\_\_(self, parent): Initializes the plot.**

**plot(self, data): Plots the given data.**

**ArduinoData: This class represents the Arduino device. It contains the following functions:**

**\_\_init\_\_(self, port, baud\_rate): Initializes the Arduino device with the specified port and baud rate.**

**read\_data(self): Reads data from the Arduino device.**

* 1. **Algorithm Details**

**The primary algorithm used in the application is the data collection loop in the ArduinoData class. It continuously reads data from the Arduino device until it's stopped. The data is then processed and sent to the GraphWindow to be displayed on the plot.**

* 1. **Local Variables and Data Structures**

**The application uses various local variables to store the state of the GUI and the data received from the Arduino device. The primary data structure used is a list to store the data points for the plot.**

* 1. **Shared (Global) Data**

**The data collected from the Arduino device is shared between the Worker, AppWindow, and GraphWindow classes. This data is accessed and modified in a thread-safe manner to prevent race conditions.**

* 1. **Dependencies with Other Modules**

**The Worker class is dependent on the ArduinoDevice class to read data from the Arduino device. The AppWindow and GraphWindow classes depend on the ArduinoData class to receive the data for display.**