Documentation DebugServer and Logger

Introduction

This project was created as a university project at the University of Applied Sciences Leipzig as part of the module “Software Engineering for Embedded Systems” by Prof. Wagner. The project group consisted of Tim Dietrich and Felix Herrling.

Purpose and Goals

The purpose of this project is to provide a web-based platform for developers to have better insight into their development when writing programs for the Dezibot. Until now, there has been no good way to actually debug what the robot is doing once it runs. Desired strings can of course be displayed on the screen, but the limited size and resolution makes this a mediocre choice for larger log messages or quantities.
In order to alleviate this issue, we have built a web-based application that runs in the existing code, which provides the developer with a website that not only shows all log entries but can also display any desired sensor data in real time. The real-time display is further supported by charts to help visualize the history of said values.
Because the Dezibot is meant to be used as a standalone tool that should not rely on outside infrastructure, we have built the web-server in its entirety on the existing code. This means that the only requirement for using this utility is a device that can connect to the Wi-Fi network provided by the robot, as well as enter the webpage via a browser.

Using the tool

In order to use the tool, the user needs to start the web-server setup by calling “dezibot.debugServer.setup();” in their code. Once this has been done, functionality can be used as described further below.
In order to access the webpage, the configuration in the “DebugServer.cpp” class can be modified. The default configuration provides a Wi-Fi network with these credentials:

- SSID: Debug-Server
- Password: PW4studProj

Once connected to the Dezibots wifi network, the webpages can be accessed by opening “192.168.1.1” in a web browser. This will lead to the main page.

Logger

The Logger Class

In order for the user (and existing functions) to interact with our logging utilities, the logger class provides an entry point to logging via various logging functions. These already determine the applied log level (for example, “logError()” will log an ERROR entry). Users need only call such a function and provide a message (string) of their choice. Timestamp and log level are saved automatically.
The Logger class follows the singleton pattern in order to ensure global access to a single object.
Additionally, log entries contain a timestamp. Because we need a timer to facilitate this, Dezibot.cpp has been expanded to call a function inside the Logger class which will in turn save the start time of the program.

LogEntry Objects

Every log object written is a LogEntry object. This struct follows a basic data form which contains:

1. Log Level (INFO, WARNING, ERROR, DEBUG, TRACE)
2. Timestamp (in absolute time starting from 00:00:00 at runtime start)
3. A message (which can contain any desired string)

Pre-built loggers

The existing Dezibot code has been modified to contain default loggers (that log at the INFO level) at most of the common functions that a user will call. For example: calling “dezibot.motion.stop()” will automatically log an INFO level log entry with the message “Motion stopped”.
The purpose of these pre-built loggers is to take load off of the developer, as they do not need to build a logger for these commonly used functions themselves. In these cases, simply running the program will already log all of their function calls.

Usage

In order to write custom logs, users will need to get the singleton instance of our Logger. This can be done by simply calling “Logger::getInstance()”. This instance can then be used to log an entry at the desired level, for example: “Logger::getInstance().logInfo(“This is a log message”);”

LogDatabase

Core functionality

The LogDatabase is the core of our logging functionality. It utilizes a singleton pattern to ensure correct usage and global availability. This means that an instance of this class can only be initialized once and will persist throughout the runtime of the program.

Furthermore, we guarantee thread safety by utilizing mutexes when writing log entries. In principle this makes sure that operations are run on a single thread which keeps us safe from multiple logger calls colliding, which could lead to data corruption, race conditions or worse. This class should generally not be modified by developers, as it only handles the log entries.

Reading logs

The web-server needs to be able to read log entries from our database. In order to facilitate this, we provide 2 different functions that provide a vector of logs. Once again we utilize C++ Mutex functionality to ensure that the reading of our log entries does not run into race conditions or similar issues.
Because it is impractical to send the full log to a user on every fetch request, we instead only do this when the webpage is opened for the first time. In that case, all logs are sent to the user. After this, a second function is utilized, which will only return new log entries to the web-server. This is made possible by simply keeping track of the last-sent log entry.

DebugServer

Web-Server

The DebugServer component implements a basic web server that runs locally on the Dezibot, without the need of external hardware. The ip configuration as well as the Wi-Fi credentials are defined in the DebugServer class and can be changed individually.

Webpages

There are currently 4 webpages implemented.

The main page serves as a central hub to navigate to the other pages.

The settings page allows the user to enable or disable sensors for the live data page. This can be done by clicking the switch next to each sensor function. This page could also be extended to include more customizability in a future update.

The live data page sends periodical requests to the web server to fetch the data for all sensors that are enabled in the settings page. The data received is then dynamically rendered into a graph and displayed to the user. The user is also allowed to change the timespan of saved data points via a slider at the top of the page.

The logging page is used to display log entries from the log database to the user. All logs are fetched when the page is initially loaded. After that, the client fetches new logs automatically every second. The user can filter the displayed logs via a dropdown menu. Please note: Writing logs is disabled while fetching sensor values for the live data page, in order to prevent spamming the log database.

File management

In order to provide easier access to the needed files for each webpage, this project utilizes the SPIFFS file system on the ESP32 chip. All the files are stored under /data and are separated into html, css and js folders. The contents of these files are streamed to the web client via a specified handler when requested. Streaming is implemented in order to avoid issues in regard to size limitations. Please note: Changes in these files are not automatically applied when uploading the C++ code to the Dezibot. Instead, you have to update the filesystem image manually.

Sensors as objects

In order to dynamically display the settings page and live data page, all existing sensors and their corresponding sensor functions are implemented as objects and initialized in the DebugServer class. All web pages that require sensor data, are rendered using these objects. Should a new sensor be added to the Dezibot project, all you have to do to update the web pages is initialize a new sensor object.