QED/HW Externship - Sensor Visualizer Report

Initial Goals:

- Visualization software for time-series data
- Capable of showing multiple streams of data simultaneously
- Support for data points and video footage
- Simple to configure and deploy

Final Product

- Web-based data visualization client (Grafana)
- Supports time series data points, video footage, geotagged data points
- Data stored locally using InfluxDB
- Automatic installation using Python, Kubernetes

Retrospective

- Our completed "MVP" for this project accomplishes most of the goals set out in our initial meetings. It can process multiple sources of time-series data and display it together through one unified Grafana dashboard.
- The final product does work, although we are having some configuration problems. Occasionally, when installing it, something goes wrong under the hood with InfluxDB that prevents it from linking with Grafana. This popped up at the tail end of the project and we aren't sure exactly what causes it, as the issue seems to occur at random. We are using the experimental Flux Query Language for Influx, which is still in beta, so it is possible that this is just a bug with the current release.
- When arriving at the final tech stack used for this project (Telegraf, InfluxDB, Grafana, Kubernetes), we researched a number of different options, from the more abstract ways of displaying data to basic diagram packages for JS. Ultimately, this stack was the best option in the given time frame. Although Grafana and Influx gave us a lot of trouble since they are designed more for real-time data monitoring across systems, they worked out of the box in terms of UX and data processing. With only a few months, we likely wouldn't have been able to create anything close to what they already can do, even if it was more specialized for this application.
- While developing this project, we encountered a number of issues, which can be mostly condensed into three main problems:
    - **Managing Kubernetes/Minikube**: Prior to this project, neither of us had used Kubernetes, so we had to learn how to use it before starting any other work. The issue was that, in order to make this system automated, we had to use Python to manage our cluster as part of the setup process. This required a fair bit of fiddling around with various commands and forum posts to understand how to properly set it up, then running the commands through Python's OS module to call shell commands directly. There were some pre existing libraries to give higher level control over Kubernetes with Python, but all the ones that we tried had compatibility issues on both our systems.

- **Automating the setup of Influx and Grafana**: Both pieces of software were designed to be manually configured by the user through their client applications. This meant that we had to work with two separate and somewhat clunky APIs to configure them individually and automatically. While it did end up working, the automated setup for each differed enough from the manual setup that we effectively had to learn how to make the project work in two very different ways.
- **Formatting data for Influx**: InfluxDB uses its own header system for tagging time series data. Since we can't expect a user to have data formatted to that standard, we had to write a basic parser to convert .csv files to an Influx-readable stream of data. There are some existing solutions that do this conversion, but none of them support the latest Influx 2.0 release. By the time we realized that, we were too deep in the 2.0 configurations to turn back.

Next Steps

- To continue this project, we would first roll back to an older version of InfluxDB, probably 1.8. This may solve the installation bug and should make parsing data easier, since we can use some of the open-source parses already on GitHub. However, it would require a rewrite of certain parts of our installation script since the 2.0 and 1.x CLIs are completely different.
- Beyond that, we would want to test the system's real-time capabilities. Up until now, we have only been using pre-existing data, but since this would potentially be used for medical sensors that feed live data, we would want to simulate that and add support. During the final weeks of the project, we discussed using an MQTT simulator to do just that, so we would continue down that route.