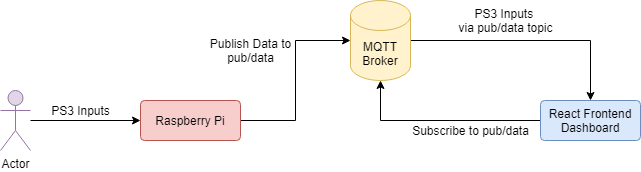
PS3-MQTT Design Document

Koi Stephanos

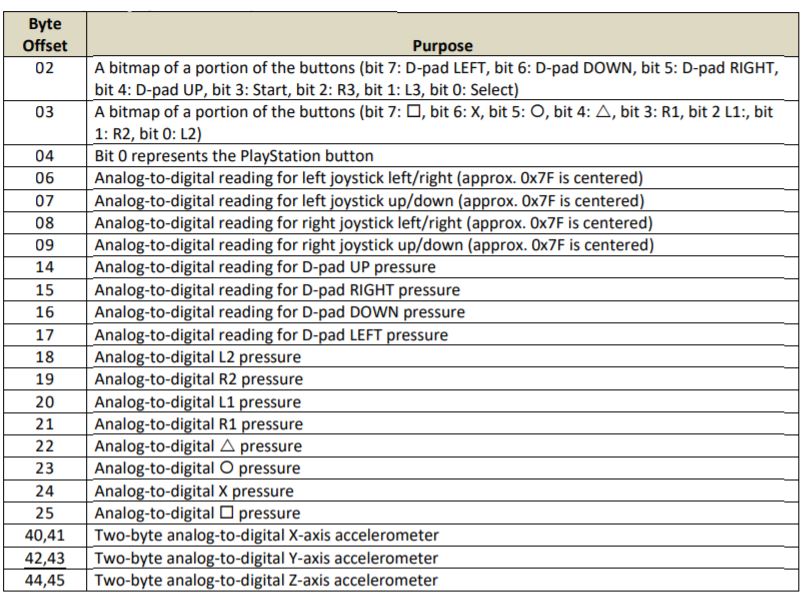
# Overall System Design:

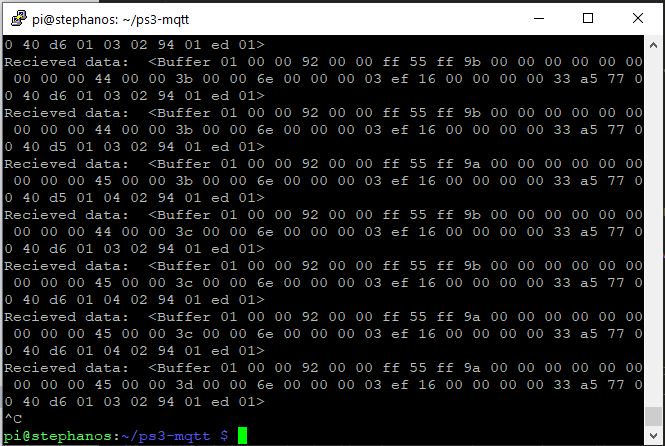
The system itself is composed of three main entities: the raspberry pi client, the MQTT broker, and the React dashboard. The pi client is responsible for reading in the binary inputs from the USB connection to the PS3 controller, the MQTT broker is tasked with managing subscriptions and distributing publish messages, and the React dashboard handles the visual and graphical presentation of the received data and controller state. A high-level diagram of the composed system can be seen below:



# USB Device Details:

This project utilizes a PS3 controller connected via USB to a raspberry pi for its binary and analog inputs. The full data being read includes all the binary values for buttons on the controller, the analog values for joystick alignment, button pressure, and accelerometer readings. For this implementation, we focus only on a subset of the binary button and the analog joysticks readings. The binary readings are set as flags, while the analog readings are decimal values associated with a range for both the vertical and horizontal position of the joysticks. The controller used to capture data in this example has the vertical center mapped to ~135 and the horizontal center to ~125. Below is a table containing an overview of the raw data, separated by offset into the message:



An example of the raw data as it is being read off the buffer is provided below: 

# List of Node Packages:

Both the client and dashboard run in a node environment, simplifying the number of dependencies present in this project. Below is a list of the most essential libraries:

## node-hid

The node-hid library provides an interface to the controller via USB. This allows us to create a device instance by specifying the vendor and product ID of our device, in this case a PS3 controller. Once we have the device instance, we can trigger a read from the USB port with a specified callback function. We write this callback function ourselves, and it is called each time data is sent from the PS3 controller, which occurs many times per second. It is within the callback function that we marshal the raw data, publish it to the MQTT broker and then initiate the next read. Some care must be taken to ensure we only publish messages when data has changed, otherwise we will quickly overflood the broker.

## mqtt

The mqtt library enables us to interact with the Mosquito broker via connect, subscribe and publish commands. Both the raspberry pi client and React dashboard utilize this library. The pi client integrates mqtt with the node-hid library in order to publish the data read from the controller, and the React dashboard uses a custom mqtt callback in order to manipulate internal state, which is then injected into the presentational components.

## react

The React dashboard is created in react via the ‘create-react-app’ [[1]](#footnote-1) library. This initializes a basic project structure with a node server and app that is injected into the browser dom. We can then add presentational components that have their state injected via a top-level container, which is connected to the MQTT broker. The development server can be started with the command: ‘npm start’.

# MQTT Topic Details:

## pub/data

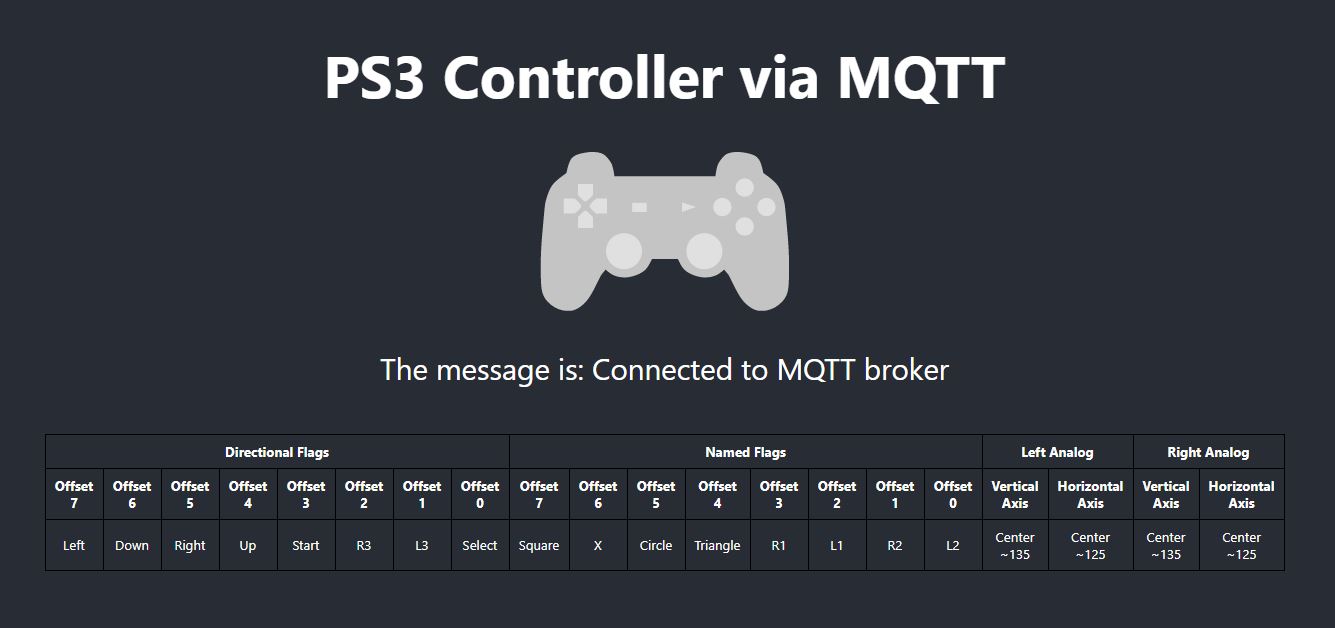
For this project, a single topic was used to coordinate messaging between the raspberry pi client and the React dashboard. A Mosquito broker was responsible for handling subscription and publish requests. To use mqtt in a browser, Mosquito had to be configured to listen to both raw mqtt on one port and WebSocket mqtt on another.

# JavaScript Source Code:

The entirety of the source code, as well as a copy of this design document, have been maintained on a GitHub repository available here: [GitHub Repo](https://github.com/dkStephanos/ps3-mqtt). Included are step-by-step instructions for setting up and launching the dashboard app.

# Dashboard:

The dashboard implemented for this project contains three main components: the incoming message, a graphical state representation of the controller, and a bitwise key for the raw data. The dashboard is written in JavaScript using the React library running on node. This allows us to use the same mqtt library to connect to the broker, albeit with a slight modification. The browser itself will not permit raw TCP connections, so we must use a WebSocket in order to connect to the broker. Upon a successful connection to the broker, the dashboard will appear like this:



Once service on the raspberry pi has connected and the controller has been turned on, the dashboard will start displaying the last received message published from the raspberry pi via the broker. The message itself contains the bitwise flags for the directional buttons as well as Start/R3/L3/Select, the named buttons, the axis readings for the left analog stick and the axis readings for the right analog stick, delimited by colons. The message mirrors the construction of the key, as seen below:



As the user presses the buttons on the controller, the binary flags are set in the message, and the corresponding buttons on the controller graphic are highlighted in blue[[2]](#footnote-2) as seen below:



Lastly, as the user manipulates the analog sticks on the controller, the axis readings are encoded at the end of the message, reduced to their approximate positions, and passed of to the controller graphic. The graphic itself highlights the analogs blue like it did for the buttons, as well as moves the analog stick to the relative position as calcluated from the message data, as seen below:



1. An overview of how to set up this environment can be found [here](https://reactjs.org/docs/create-a-new-react-app.html) [↑](#footnote-ref-1)
2. for this implementation L1-3 & R1-3 have been omitted from the graphic, and as result, are present in the message string only [↑](#footnote-ref-2)