Notes

# Software

During development of the system, the software used moved through multiple redesigns in both function and interface.

In order to avoid issues of cross-platform portability, a web application was developed as the control system – rather than platform-specific native applications. This avoided additional costs that would be accumulated through development licenses either during development or as a future commercial project. Furthermore, this simplified usage of the control system by only requiring the user to connect to CHAP’s ad-hoc wireless network in order to use the system, avoiding additional steps such as installation.

The fault in this setup is that it places the effectiveness and reliability of the system entirely in the hands of the user’s web browser. With browsers showing extremely mixed results in support of modern HTML5, CSS3, JavaScript and other key web technologies; browser support cannot be relied on. For example, access to a user’s webcam through WebRTC (such as for face tracking) is a feature that cannot be guaranteed between browsers. That said, the continued development and advancement of web browsers can be relied upon and isn’t prone to the same level of change or redesign visible between both different mobile operating systems as well as operating system updates.

## Prototype 1

The initial system developed as a proof-of-concept consisted of multiple sub-systems working in parallel. As seen in the later systems, a web-server, image capture system (through attached camera) and event listener system existed. These were created using the following software solutions:

* Web Server -> *Apache2*, HTTP server hosting software.
* Image Capture -> *Motion*, continuous image capture and serving system using motion detection.
* Event Listener -> *NodeJS*, JavaScript runtime engine (JavaScript software without web browser). Through the use of the Johnny-Five system, NodeJS was capable of communicating with an Arduino running Firmata (communication protocol). Socket listening was accomplished using Socket.IO.

Image streaming was slow and required very low-resolution images (320x240) for real-time transmission over ad-hoc (local) WiFi. Furthermore, the high number of sub-systems and programming languages used was felt to be inefficient. With Apache running web servers across the world, a single-client locally run website did not need such an extensive system.

Raspberry Pi

Arduino with Firmata

Web browser on any device

Motion

NodeJS listener with JonnyFive

Camera

To motors

Apache2

Figure 1 - Prototype 1; multiple languages and entirely independent systems used. Minimal internal communication, with no central system.

## Prototype 2

The second system developed aimed to minimize the number of subsystems, primarily aiming to develop a single solution using a single programming language. The ability of NodeJS to communicate directly with hardware as well a large, modular, open source library, made it a good candidate for an upgrade control system. The same key subsystems existed, however, these were combined into a single file with no outside software required other than OpenCV for image capture and processing. Client-Server communication was again managed through event-based messages using Socket.IO.

* Web Server -> *NodeJS, Express module*, HTTP web framework.
* Image Capture -> *NodeJS and OpenCV*, an event listener within the NodeJS server listened for client messages requesting image frames, this led to an OpenCV process capturing a new frame as a JPG and returning the file name for serving to the client.
* Event Listener -> *NodeJS*, JavaScript runtime engine (JavaScript software without web browser). No changes from the initial prototype other than merging the system into the server program.

This system proved reliable and much simpler than the previous one. Images were only captured once the client had successfully loaded the previous frame, this limited the frames-per-second to internet and processing speed. However, the camera was re-opened for each frame, rather than held open; to solve this the image capture system would need to be run independently (in parallel), and serve on request directly to the user. That said, high-quality streaming was achieved, albeit with noticeable latency.

Arduino with Firmata

Raspberry Pi

Web browser on any device

OpenCV executable

NodeJS Server

Listener with JonnyFive

Camera

To motors

Figure 2 - Prototype 2; while a simplified setup within the machine, more levels of communication exist. Self-contained system only reliant on OpenCV compilation and installation. Bottleneck visible, with all information moving through server.

## Final System

The need for an independent image capture system was followed by the introduction of USB2Dynamixel for direct motor control without an Arduino. With a working USB2Dynamixel library already existing in Python, Python was used for this system. Using Python’s own ability to listen on web sockets, as well as the default HTML/JavaScript web socket functionality, communication between the client and the motor controller was achieved. This system removed the server as a communication bottleneck, freeing up the server for HTTP serving only. With this accomplished, it was found that Python could serve the web page with no limitations. The *Flask* framework was used and proved highly successful with a dynamic web serving ability which allowed for inbuilt camera streaming. This change resulted in near perfect 720p live streaming over a local network.

* Web Server -> *Python* served using the *Flask* framework (similar to Express used in Prototype 2) with a continuous multi-user ability added through the use of *gevent* (networking library). The IP address of the server is passed to the client through a *Flask* variable for use with the event listener.
* Image Capture -> *Python, Flask and OpenCV*, a *Flask* variable exists within the HTML web page as an image source, on serving by *Flask* this is continuously updated with binary image data.
* Event Listener -> *Python and Simple USB2Dynamixel*, uses a web socket to receive commands from the client. This is a separate running subsystem on a preset port.

Raspberry Pi

Web browser on any device

Python Flask server

Camera

USB2Dynamixel

Python listener

Figure 3 - Final System; highly simplified. Minimal systems, minimal communication with no bottleneck effect. OpenCV used but integrated. Arduino replaced with direct USB2Dynamixel communication.

# Hardware