Progress Report #2

Work Done:

1. **Create list of requirements for server side software**
    a. Send data from cameras and low level sensors to a remote client
    b. Receive data from remote client and use to control low level actuators
    c. To do a) and b) as efficiently as possible
2. **Investigate server side software for controlling hardware via HTTP requests**
    a. In group meetings on 2013-08-01 and 2013-08-05, agreed on a web based approach
    b. Investigate CGI script approach (under apache2)
        i. Sensors/Actuators/Video run in separate process(es)
        ii. Client makes HTTP request handled by apache2; apache2 starts a CGI program
        iii. Process for using CGI with apache2: http://httpd.apache.org/docs/current/howto/cgi.html
        iv. CGI program must communicate with sensors/actuator/video program(s) using unix domain sockets or named pipes. It then prints results which apache2 relays back to the client.
        v. This approach requires interprocess communication techniques to be included in all programs. Although it allows us to use an existing HTTP server (apache2), it is likely more work than writing a custom HTTP server to integrate all the systems.
        vi. Interprocess communication is generally inefficient, as separate programs have separate memory and must transfer data between each other at agreed times. There is possibility for deadlocks.
    c. Investigate a custom HTTP server approach
        i. Sensors/Actuators/Video run in same process as HTTP server, under separate threads
        ii. HTTP defined in RFC2616 standard: http://www.w3.org/Protocols/rfc2616/rfc2616.html
        iii. Respond to GET requests to give data to client, and POST requests to control actuators
        iv. Threads running within the same process share memory, which makes communication simpler and more efficient. However there is still a possibility for programming errors to cause deadlocks or crashes. Significant testing and debugging time will be needed.
3. **Implement proof of concept HTTP server**
    a. Written in POSIX C99 but can be easily ported to C++. Available at https://github/szmoore/MCTX3420
    b. Can respond to GET and POST requests according to HTTP/1.1
        i. Tested with modern web browsers (firefox/chrome)
    c. Can transfer HTML and JavaScript files that allow web browser to produce a GUI
    d. Can respond to a specific GET request by returning simulated data, instead of a static web page.
    e. Tested on Raspberry Pi.
    f. Plotted performance using JavaScript flot library.
        i. Found JavaScript execution times ~20ms to produce line graph from localhost
        ii. Server side execution times ~1us.
4. **Communicating with other teams**
    a. Verbally told people from pneumatics, sensors and electronics teams that we were expecting to use a Raspberry Pi.
        i. Either communicating with another device over USB, or using GPIO to control other devices
    b. Responded to email from Coen (Electronics team) about use of Raspberry Pi as microcontroller and communication protocols for sensors/actuators. Currently organising a meeting.

TODO:

1. **Confirm server hardware with other teams**
2. **Collaboration within team**
    a. Decide on common language (C/C++ or Python) with Justin/Rowan and Callum
        i. A multithreaded server will need a common language
    b. Consider how data will be stored server side and transferred to the client in more detail with Justin/Rowan and Callum
    c. Consider what hardware interfaces will be needed with Justin/Rowan
    d. Consider HTTP interface presented to client with James
3. **Start to develop framework for multithreaded server side software**
    a. Expand proof of concept server to a multithreaded server, using dummy functions.