MCTX3420 Team 4: Progress Report #4 (Summary)

Sam Moore, Rowan Heinrich, Callum Schofield, James Rosher, Justin Kruger, Jeremy Tan

Callum:

1. **Fixed compiling errors in regards to compiling OpenCV code and accessing header files and libraries**
2. **Wrote the pseudo code for capturing and storing an image in OpenCV out into C**
    a. Debugged and modified code to fix some errors
    b. Tested code, works with laptop camera however is only able to save an image roughly once per second
3. **Added code to timestamp the images**

Sam:

1. **Implement Sensor handler function for FastCGI**
    a. Merged Jeremy's FastCGI API with the server framework
    b. Wrote a Sensor Handler function to respond to HTTP requests with sensor data.
2. **Improve multithreaded framework's error handling and exit condition checking**
    a. Create run state that must be checked periodically by all threads, set (once only) by any thread on exit.
    b. Difficulties getting FastCGI request loop to exit, since *FCGI_Accept* is a blocking function
    c. Running server in valgrind to find memory errors. See http://valgrind.org/
3. **Consider transfer of sensor data in more detail**
    a. Double buffer tested by Jeremy is fast; could be problems if multiple requests arrive at the same time.
    b. Time stamp sensor data using *gettimeofday(2)*. Work on transferring data acquired since a specified timestamp (instead of just dumping a buffer of the most recent points). Use *clock_gettime(2)* instead?

Jeremy:

1. **Completed integration of FastCGI code with the main server code**
    a. Reworked the exposed functions in fastcgi.c to make more sense and for convenience
    b. Status reworked into the JSON reply instead of the HTTP status code to overcome AJAX limitations
2. **Tested authorization scheme to oversee who has control of the device at any one time**
    a. Authorized users must enter user/pass to gain an access token
    b. Access token controls who has control at that point
3. **Explored double buffer concept for obtaining sensor data**
    a. Potentially faster (mutex only covers pointer swap)
    b. Concurrent access may/may not be an issue; benefit over binary file questionable

James:

1. **Wrote a dummy test UI page**
    a. UI test page to test client to server interactions.
2. **Started writing rules for handling of data**
    a. 2 priority levels. High and Normal.

Rowan & Justin:

1. **Investigated coding on the BeagleBone Black (BBB) and system capabilities:**
    a. Three languages: Python, Javascript and C; currently pursuing C code to attach with server
    b. ADCs and pins can be interfaced through Linux sysfs e.g. /sys/devices/platform/tsc/ain#
    c. Testing commands can be sent through Linux shell e.g. cat # /sys/class/gpio/gpio%d/attribute
    d. Online documentation resources for interface: circuitco.com/support/index.php?title=BoneScript
2. **Coded sample programs to interact with GPIOs and control systems on BBB:**
    a. One sample (C) that triggers and reads a sensor attached to one of the ADC modules
    b. One sample (C) to blink an onboard LED and send commands to GPIO pins
    c. Modified BBB inbuilt generic_buffer.c drivers, can now read and write blocks of sensor data

Information exchanged with other teams (Meeting 2013-08-20 at 9am):

1. Two cans will be used. At least one camera will be used with no processing (stream images).
2. Another (or the same) camera *might* be used to process images from an interferometer.
3. Beaglebone will be used. It has already been ordered.
4. Wheatstone bridges will be used, so a single ADC will be used to read multiple sensors
5. Told electronics team not to worry about writing software on the Beaglebone, since that's our job.

Work TODO:

1. **Get basic GUI implemented for testing how data is transferred through the API**
2. **Add Actuator Handlers and sensors interface functions to server**
3. **Test code on an actual Beaglebone**
4. **Investigate streaming of images using Beaglebone**