CS Capstone Mid-Spring Progress Report

May 6, 2018

Project LOOM

Prepared for

Kevin McGrath
Kirsten Winters

Prepared by

Group 36

Trevor Swope
William Selbie
Luke Goertzen

Abstract

This document is a progress report of Senior Capstone Group 36's progress for the first half of Spring Term, shortly before Expo. The report briefly reiterates the purpose and goals of the project before presenting the its current status. The details, problems, and solutions involved in reaching this current state are divided and enumerated by each member.

CONTENTS

LIST OF FIGURES

LIST OF LISTINGS

1 Recap of Project Purpose and Goals

With Project LOOM, we aim to create an open-source, plug-and-play, suite of modular building blocks, the extensible and easy programmability of which expands the demographic of people capable of implementing Internet of Things solutions. For users with limited technical expertise to create complex systems, we aim to build a system that abstracts out the more technical details, allowing them to focus on their system more than the implementation of the modules. The system should also be usable by higher level students and experts by allowing them to modify or write their own firmware, and create new modules.

2 Current Project Status

2.1 Hardware

At this point, we are done adding new devices, actuators, and sensors (further additions would just be for our client) and are currently consolidating all of the existing code for devices into the LOOM library, testing to make sure different combinations of components work together. The only exception to this is making sure we have enough of each I2C supported sensor to fully test the MuxShield with a variety of permutations of attached sensors.

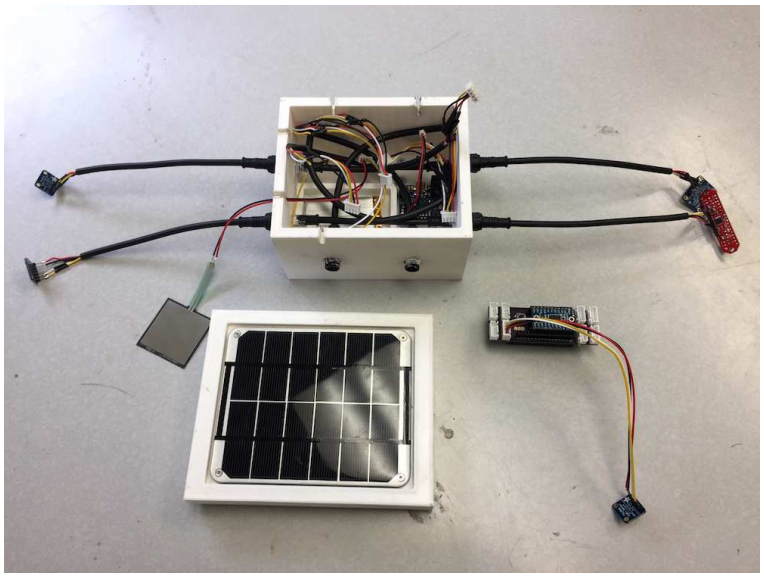


Fig. 1: Multiplexer and Enclosure

2.2 LOOM Library

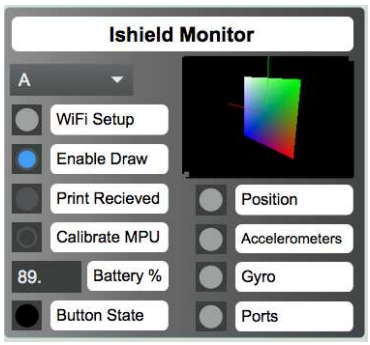
As the code base for Project LOOM expanded in size and complexity, the organization of the code was due for a redesign. That which had become spread out between many different Arduino sketches, each with their own extra files of configurations or subroutines, has been consolidated into a modular and dynamic LOOM library. There is now a single master sketch and a collection of files for components, such as communication platforms, sensors, and common functions. When a user compiles their sketch, preprocessor statements will include only the necessary code, based on the configuration file.

Additionally, the library architecture makes it easy for similar functionality with different underlying technologies to be handled in the same manner. For example whether the user is uploading their code to a Feather M0 or 32u4 makes relatively little difference from the user's view. The correct functions are called without any changes needing to be made by the user, as the preprocessor selects only the relevant code for the selected options. Similar abstractions apply to communication platforms, e.g. WiFi and LoRa.

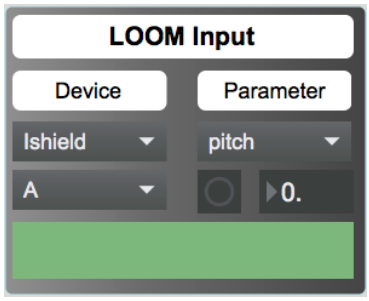
The LOOM Library is designed to be easily navigated, modified, and extended by further users. Its modular nature means that users can remove unnecessary components (though the preprocessor would also do that before device upload) or add new components by filling out a template file. The end result of this architecture is an easily followed paradigm for code structure and distribution of functionality that will allow the library's code base to grow cleanly and sustainably even once the original developers have left.

2.3 Channel Interfaces

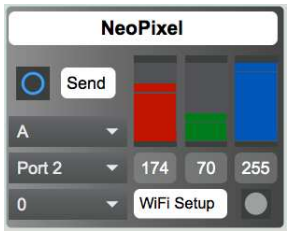
We now have Max interfaces including Neopixel, relay, and servo controllers; LOOMin general receiving module, Ishield monitor, I2C multiplexer module; and a channel manager (on top of all of sending, receiving, and processing modules made by Chet, or whatever else the user makes from scratch). Each of the Max modules has a full detail version (expert mode) and a cleaner, simpler version that abstracts away a number of details (channel mode). The channels hide device instance numbers, IP addresses, and UDP ports behind a single channel, such as A, B, C, etc. The channel manager can detect when a new device is added to the network and can assign it the next available channel.



(a) Ishield Monitor



(b) LOOM In Sensor Viewer

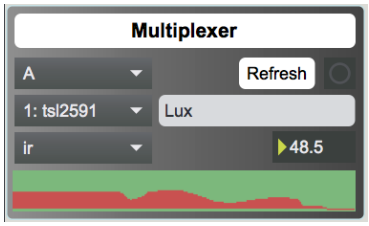


(c) Neopixel Controller

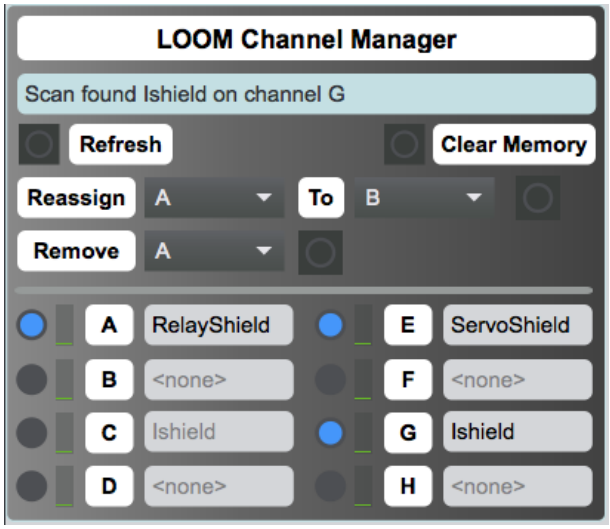
(d) Servo Controller

Fig. 2: Max/MSP Control Module Interfaces

(a) Relay Controller



(b) Multiplexer Monitor



(c) Channel Manager

Fig. 3: More Max/MSP Control Module Interfaces

2.4 Adafruit IO and IFTTT

We have successfully demonstrated being able to use the external services Adafruit IO and IFTTT. The former is an MQTT based platform that can send and receive from devices on the network, and can perform limited actions based on conditional triggers. The LOOM network applications can easily be expanded by linking to IFTTT through Adafruit IO. The popular platform for chaining actions based on conditions allows users to easily get output from their network in a variety of ways.

The code we have used for this currently stands by itself and will be incorporated into the LOOM library shortly. Further, we would like to be able to use IFTTT to send data into a network, rather than only acting on data sent out from the network. This, however, is more of a stretch goal and it is not imperative that it gets implemented.

2.5 Demos

We are also preparing demos for Expo, having Chet, rather than us, lead the demo. We can verbally explain how to do something, but he will have control of the computer. This ensures that not only can we clearly explain how to do certain things, but also that our interfaces and functionality are inherently understandable to the point of minimal explanation. The details of these demos are discussed in further detail in the interface user study section below.

2.6 LB Farms

LB Farms, a local farm owned by university professors, is where the first deployment of Project LOOM took place. Four Decagon sensors split across three development boards, two evaporimeters, and an ethernet and LoRa enabled hub were deployed there. The Decagon sensors transmit every 15 minutes while the evaporimeters transmit every 10. The data is transmitted through LoRa to the hub, and then the hub's processor forms the data into the url arguments for a GET request, which is sent to PushingBox via ethernet. A Google Apps script on PushingBox then makes a GET request to a Google Sheet, which records the data.

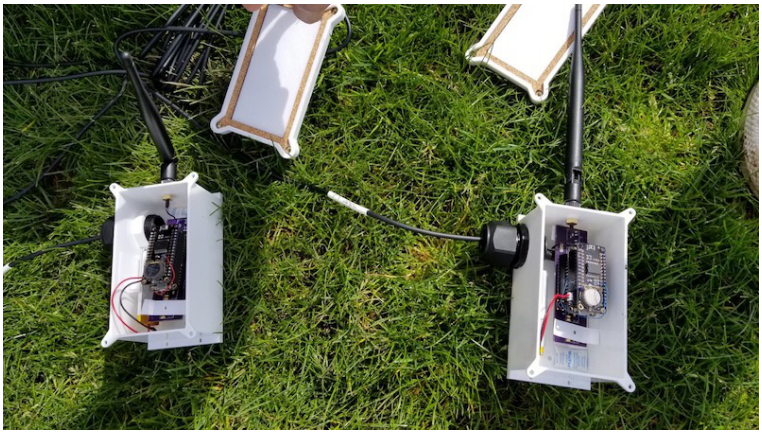


Fig. 4: Two Decagon Sensors at LB Farms

3 What We Have Left To Do

3.1 LOOM Library

While we have put a lot of work into getting the library to the state it is currently, there are still a few small pieces of code that are not yet in the library that need to be added. We also continue to improve the commenting and organization as we work on it. Beyond this, while the bulk of the functionality is contained within the library, it will be important moving forward to make sure the structure and distribution of functionality is consistent and clean so that moving forward it is easy to expand upon.

3.2 Documentation

We also want to improve the documentation to a more final and formalized state, suitable for future users of Project LOOM and the associated library. This documentation will clearly explain the components of Project LOOM and how to setup and use them. Additionally, we will include readme's to the library and documentation on how to add to it.

3.3 Scripts and IDE Bypasses

We intend to have a command line script that can generate a configuration file for the user, and then also be able to upload the sketch (created dynamically based on that config) from the command line, without needing to open the

Arduino IDE. Once we have these scripts, we will probably be able to run them from within Max, via a graphical interface for selecting configuration options and a button to compile and upload to the device.

3.4 Design Document

Lastly, we need to update our design document with any modifications or refinements that have been made to the Project LOOM design. Most notably, the document as it stands now lacks any mention of the decision to implement a LOOM library.

4 Problems We Had

Automating the config creation, firmware compiling and upload is a bit tricky without the Arduino IDE, as it does not show all the necessary commands to run the process for the command line. Ideally we will have this before running before Expo, but it may have to be deferred until afterwards.

Another issue is the constraints of programming for the Feather 32u4. With less programmable flash than the Feather M0 (32KB for the 32u4 vs 256KB for the M0) we've had to cut down on some functionality when building for the 32u4.

The M0 standby() does not function properly when connected to a Serial USB; this has given us trouble putting the M0 boards into true low power mode.The current solution has been to manually disable the Serial hardware, but it makes it incredibly difficult to debug without the serial monitor.

5 Interesting Pieces of Code

Here we have a truncated piece of the multiplexer code, which checks the ports for the type of device connected and measures and packages that data accordingly. This packaged data is stored in a bundle which is sent after this function returns.

```
void get_sensor_data(uint8_t i2c_addr, OSCBundle *bndl, char packet_header_string[], uint8_t port)
{
    #if LOOM_DEBUG == 1
        Serial.print("Attempting to measure data from sensor with address: ");
        Serial.println(i2c_addr);
    #endif
    #ifdef i2c_addr_tsl2591
        if(i2c_addr == 0x29){
            if (setup_tsl2591()) {
                measure_tsl2591();
                package_tsl2591(bndl,packet_header_string,port);
                return;
            }
        }
    #endif
    #ifdef i2c_addr_fxos8700
```

```
        if((i2c_addr == 0x1C) || (i2c_addr == 0x1D) || (i2c_addr == 0x1E) || (i2c_addr == 0x1F)){
            if (setup_fxos8700()) {
                measure_fxos8700();
                package_fxos8700(bndl,packet_header_string,port);
                return;
            }
        }
    #endif
    ...
    other sensors in the same format
    ...
    #if LOOM_DEBUG == 1
    if (i2c_addr != 0x00) //sht31d hardware bug
        Serial.println("This sensor is not currently supported by the Project LOOM sytem");
    #endif
}
```

Listing 1: Multiplexer Get_Sensor_Data Function

The message router is used to route OSC messages to the correct function to handle them.

```
void msg_router(OSCMessage &msg, int addrOffset) {
    #if LOOM_DEBUG == 1
        char buffer[100];
        msg.getAddress(buffer, addrOffset);
        Serial.print("Parsed ");
        Serial.println(buffer);
    #endif

    #if is_tca9548a
        if (msg.fullMatch("/GetSensors", addrOffset)){
            msg.add(configuration.packet_header_string);
            #if LOOM_DEBUG == 1
                Serial.println("Got a request for sensor list");
        #endif
        }
        msg.dispatch("/GetSensors",   send_sensor_list, addrOffset);
    #endif

    #if num_servos > 0
        msg.dispatch("/Servo/Set",    set_servo,      addrOffset);
    #endif
    #if is_relay == 1
        msg.dispatch("/Relay/State",  handleRelay,    addrOffset);
    #endif
    #if is_mpu6050 == 1
        msg.dispatch("/MPU6050/cal",  calMPU6050_OSC, addrOffset);
    #endif
    #if is_neopixel == 1
        msg.dispatch("/Neopixel",     setColor,       addrOffset);
```

```
    #endif

    #if is_wifi == 1
        msg.dispatch("/Connect/SSID",     set_ssid,     addrOffset);
        msg.dispatch("/Connect/Password", set_pass,     addrOffset);
        msg.dispatch("/wifiSetup/AP",     switch_to_AP, addrOffset);
        msg.dispatch("/SetPort",          set_port,     addrOffset);
        msg.dispatch("/requestIP",        broadcastIP,  addrOffset);
        msg.dispatch("/getNewChannel",    new_channel,  addrOffset);
    #endif

    msg.dispatch("/SetID", set_instance_num, addrOffset);
}
```

Listing 2: Message Router

5.1 LOOM Library

As mentioned previously, the LOOM Library is the answer to the growing complexity of managing all of the variety of functionality Project LOOM supports. The library is interesting in that one can build a sketch to their specifications just by modifying the configuration file. The preprocessor will select the code needed to support the specification. The user does not have to go change which functions are called when switching something like hardware or communication method. Users can however, modify or add to the library to get customized or further functionality than the base library. This code in question

5.2 Channel Manager

The channel manager is an interesting, mostly passive, Max patch that, if open, can wait for new LOOM devices to be added to the network, and assign them available channels accordingly. Once a device has a channel, a user can easily set the channel in another Max patch to communicate with that device (assuming the patch is for the correct type of device), automatically configuring the instance number and port needed to receive from the device, and requesting the its IP address so that messages can be targeted rather than broadcasted to all devices.

6 Includes description of results from user studies

6.1 Last term demos

Last term, we had demos of the current state of our project at the open house of the lab and the IoT workshop. These were important to our learning what needed to be changed for smoother demos and simpler, easier to follow interfaces. While the open house demos were led by us, and the workshop was largely guided by us, the expo draft demos (discussed next) should be understandable and executable to the users' ideas. This was a large part of the reason for implementing the channel based interfaces, minimizing complexity and room for error on the part of the testing user.

6.2 Expo draft demos

Given that all three member of the Computer Science team are working upon varied and occasionally overlapping aspects of the LOOM Library, it can be difficult to maintain consistency of demos, since the set of firmware flashed to the device can change even without the configuration file changing depending upon who uploads the firmware. Also, some of the functionality that we would like to demonstrate, such as the Google Sheet that is routinely updated by the LoRa Hub on LB Farms, is reliant upon having a stable internet connection, which is not necessarily a guarantee at Expo. Despite these challenges, we expect to have some very robust demos by Expo.

7 Conclusion

Progress thus far this term has been good, we have just a few things left that need to be done before Expo. We also have a few refinements and improvements that we can still work on for Chet after the code freeze and Expo in a more leisurely manner, mostly regarding documentation and organizations (functionality will stay more or less the same). We have all learned a lot during the course of working on this project, and are excited to see where it goes after it is out of our hands.