Artificial Shake Mechanism -
For Freefly MōVI Pro by Arduino Esplora Microcontroller

Windy Films Internship Project Report

Benjamin Glass

July 27, 2017

Abstract

With camera movement, cinematographers seek complete control. Even in the case of camera shake, this is true. Whether a filmmaker attempts to capture the small jitters from an earthquake rumbling a building, the jostling of a car moving fast over bumpy terrain, or the slow weaves that are characteristic of a camera handheld, these frame motions need to be deliberate. With machinery, we have the ability to pinpoint the frame motion we desire, eliminating the human element – and the likely associated error.

In this report, I describe the process of developing a mechanism that adds artificial shake to a camera frame. I programmed an Arduino Esplora microcontroller to control a MōVI Pro's gimbal motors through the Gimbal Control Unit. The Arduino sends commands that change the motors rotational velocity, that result in jerky camera motion.

Windy Films treats this project as a proof of concept. Although the mechanism is functional and expandable, the hardware limitations and lack of time for production lead us to look to others for further development. By making the software available and presenting the hardware configuration, our hope is that this mechanism appears as a widely useful feature that can be integrated easily into an existing environment, including the Freefly MōVI Pro. As a motion feature incorporated into the MōVI environment, the shake mechanism would be (a) developed more extensively, and (b) the mechanism would be more easily operated.

1. Introduction

Timeline and Assignment

The goal of this summer project was to create a mechanism that adds artificially generated shake to a camera shot. As an example application, an operator wants a jittery frame when they are shooting a car chase scene over bumpy terrain from the driver's perspective. Many times, shake is used to enhance the mood of a moving image. In the case of a driver plowing over a dirt road at high speed, the cinematographer most likely wants to impart a sense of motion in the viewer. This mood would not be conveyed as well if the camera were steady.

It was important to not stray too far away from this basic overall goal of the entire mechanism, that the job of the shake device, in whatever form it manifests, is to convey a mood. Like all other tools, whether it be lights, lenses, or camera stabilizers, this shake mechanism should be a utility for the filmmaker. Hence the goal of the project was to create a device that would supply this special motion feature to an operator's toolbox.

The report is organized as follows:

In **Section 2**, I list project objectives.
In **Section 3**, I illustrate the complete prototype overview in a schematic diagram.
In **Section 4**, I describe all parts of the project hardware.
In **Section 5**, I describe all parts of the project software.
In **Section 6**, I describe limitations of the project.

2. Objectives

There were three main objectives for the mechanism.

- **Realism:** The mechanism should produce believable and natural shake. Randomness and variability need to be taken into account. Consider "dead-time" (moments of no shake).

- **Usability:** The mechanism should be easy for cinematographers to use. The hardware should not inhibit the operator's ability to shoot footage. The shake mechanism should be easy to turn on and off, as well as store. Durability should be considered.

- **Flexibility:** The mechanism should be able to be easily modified for the needs of the operator's project. Different shaking styles should be able to be accessed easily (quick mechanical jitters, bigger random surges, etc.). The software should be documented and extendable to allow for future needs.

3. Prototype Overview

Figure 1 is the schematic diagram of the mechanism.

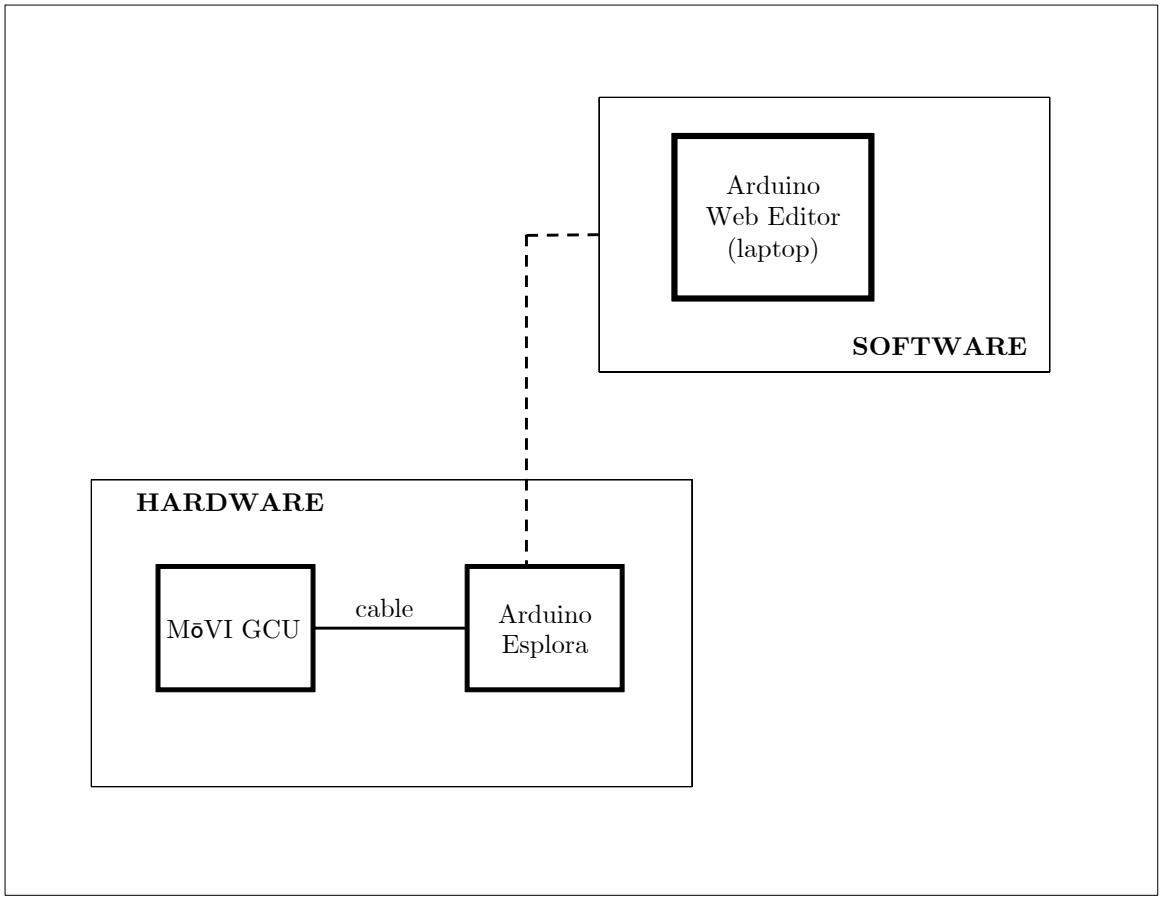


Figure 1: Schematic Diagram

Commands are written in the Arduino language and uploaded to the Arduino Esplora through the Arduino Web Editor. The Esplora controller communicates to the MōVI GCU through a cable. See **Section 4** for descriptions of the MōVI Pro, the MōVI GCU, the Arduino Esplora microcontroller, and the connection cable. See **Section 5** for a description of the Arduino Web Editor, the Freefly API, and Arduino code algorithm uploaded to the Esplora.

4. Hardware Description

The shake mechanism requires a set of hardware that controls the motion of the camera, most readily a platform to secure the camera, and a process to shake the camera.

4.1 MōVI Pro

The MōVI Pro is a *camera movement system* made by *Freefly Systems*. The MōVI[1] was optimal for this project because of its reputation as a professional filmmaking tool, its ability to securely handle a camera, and its accessible Application Programming Interface (API). **Figure 2** shows an image of the MōVI Pro Gimbal, the interior hanging harm that secures the camera, the cage, and the MōVI ring.

Figure 2: MōVI Pro Gimbal

The Gimbal Control Unit (GCU), as the computerized component of the MōVI Pro, controls gimbal and camera mechanics, including pan, tilt, roll, focus, iris, and zoom. The GCU appears as a compact box that sits along the spine of the MōVI Pro Gimbal, as shown in **Figure 3**.

[1] *"mow (e.g. mow the lawn) - vee"*

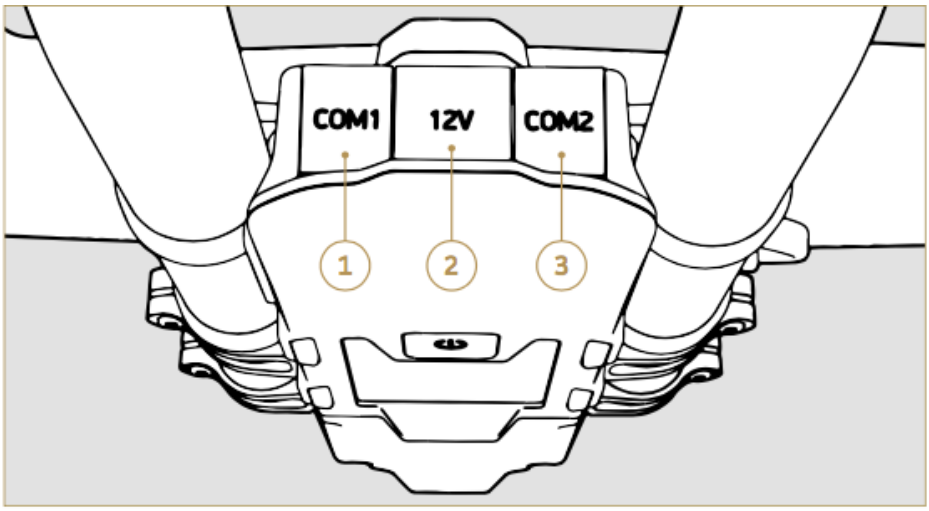


Figure 3: Gimbal Control Unit with Numbered Ports (Top-view)[2]

Like a traditional camera stabilizer, the MōVI Pro Gimbal can secure and balance a camera, allowing for smooth footage with human operation. With the addition of the GCU, the MōVI Pro can be directed and operated by use of a controller, with capability for motion control in the x, y, and z directions, as well as camera lens control (zoom, and focus). In practice, one operator holds the MōVI Pro, moving through space with the camera, while a second operator adjusts the gimbal's direction using the controller.

Although the official MōVI controller[3] is tailored directly for MōVI Pro use, Freefly Systems allows for more control capability with respect to the MōVI. In March of 2017, Freefly publically released their API, allowing connection between third party controllers and the MōVI Pro. Due to this reason, I was able to design a controller that I could develop uniquely for the project.

4.2 Arduino Esplora Hardware Controller

In order to control the MōVI with the GCU API, I needed a programmable hardware controller. The Arduino[4] microcontroller platform suited my needs. As a software-hardware package with a gentle learning curve, Arduino prides itself on being accessible to new users with little experience with electronics. Arduino's basic microcontrollers, like the *Uno* and *Esplora*, are well documented, with many tutorials and personal project examples available online.

Within the Arduino microcontroller family, I used the Arduino Esplora, for its game board features, as seen in **Figure 4**. Freefly has also documented and published Esplora-MōVI connections using the Freefly API. These documents explain how to setup an Arduino Esplora to

[2] MōVI Pro Operation Manual, page 21
[3] www.store.freeflysystems.com/products/movi-controller
[4] "Arduino...an open-source electronics platform based on easy-to-use hardware and software."

control a MōVI Pro's basic mechanics. From these manuals, I was able to understand the basics of setting up the Freefly API in the Arduino code, how to create a connection between the microcontroller and the GCU, and how to construct commands to the GCU through the API (*e.g.* how to pan a camera 3 degrees to the left).

The Esplora controller runs for around $43 on the Arduino website, where we purchased it.

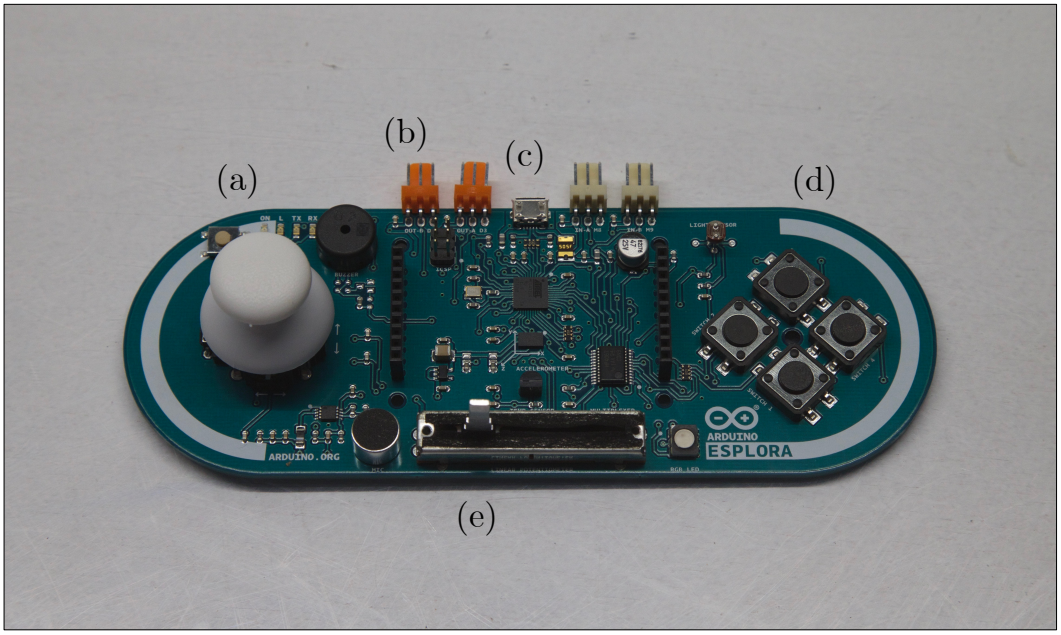


Figure 4: Arduino Esplora Microcontroller - **(a):** Arduino Joystick **(b):** 3-Pin OUT-B Port **(c):** USB port **(d):** Arduino buttons **(e):** Arduino slider

4.3 Arduino-MōVI Pro Cable

There was no existing cable to connect the Arduino Esplora to the MōVI, so I built one using simple electrical cables.

The Arduino has a three-pin OUT-B port, as labeled in **Table 1**.

+5V	**Digital Pin 11 (TX)**	**GND**
Pin 1	Pin 2	Pin 3

Table 1: OUT-B Port Pin Diagram for Arduino

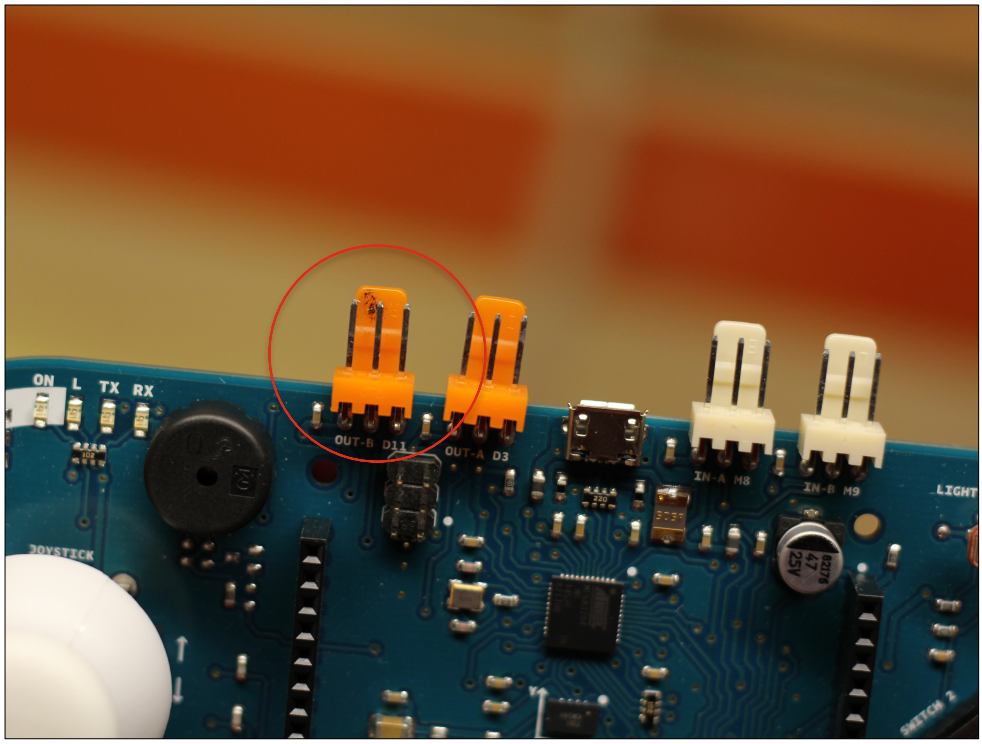
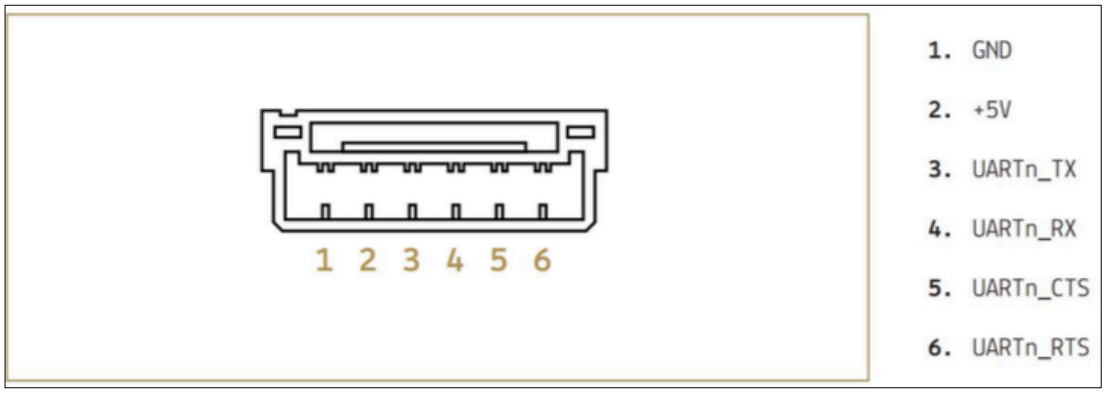


Figure 5: OUT-B Port on Arduino board, circled in red.

The MōVI Pro COM1 Port is on the left side of the GCU (see **Figure 3**). It is of type JST GH 6-PIN, and illustrated in **Figure 6**.

[5]



Fgure 6 – 6-pin diagram for MōVI COM1 Port

The cable connection is 3-pin to 3-pin. The Arduino is only responsible for sending data out (TX), and the MōVI is only responsible for receiving data (RX). Digital Pin 11 (acting as TX[6]) connects to UARTn_RX in COM1. **Figure 7** illustrates the pin connection.

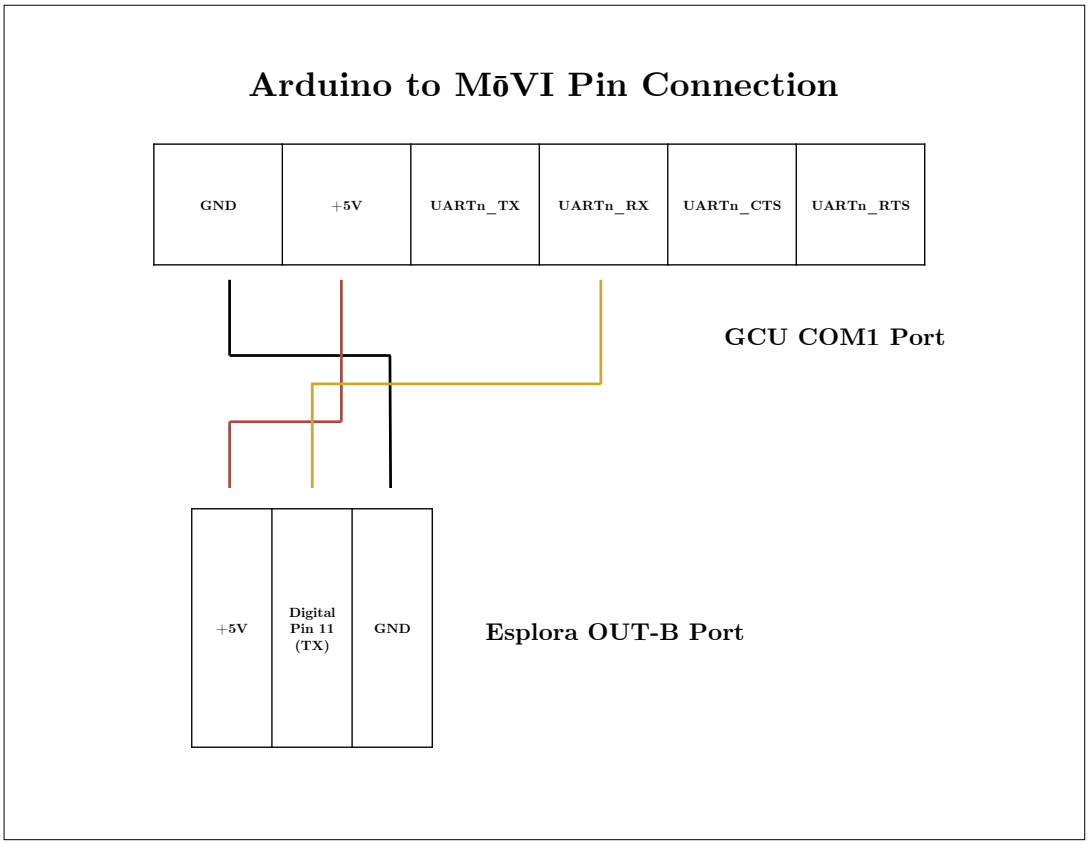


Figure 7: Arduino to MōVI Pin Connection

[5] MōVI Pro Operation Manual, page 23
[6] Software configuration in Arduino code, line 15 of file

The Arduino-MōVI cable was comprised of three separate segments of wires. I used three F-M wires[7] from a basic Elegoo Electronics kit to connect to the 3-pin OUT-B port on the Arduino.

I used a *Mōvi Pro COM to Mōvi Controller Receiver* Cable[8] to connect to the COM1 port. I stripped the far side using a wire cutter to get access to the three wires necessary for the connection (ground, 5 volts, and RX).

To connect the Arduino side wires and the MōVI side wires, I used a set of three extension wires from the same Elegoo kit (They came from a Servo motor, which I cut off from the wires). One end of the extension wires had female connectors, which I connected to each male DuPont wire (Arduino side). I stripped the other side of the extension wires and manually twisted these wires with the stripped MōVI wires.

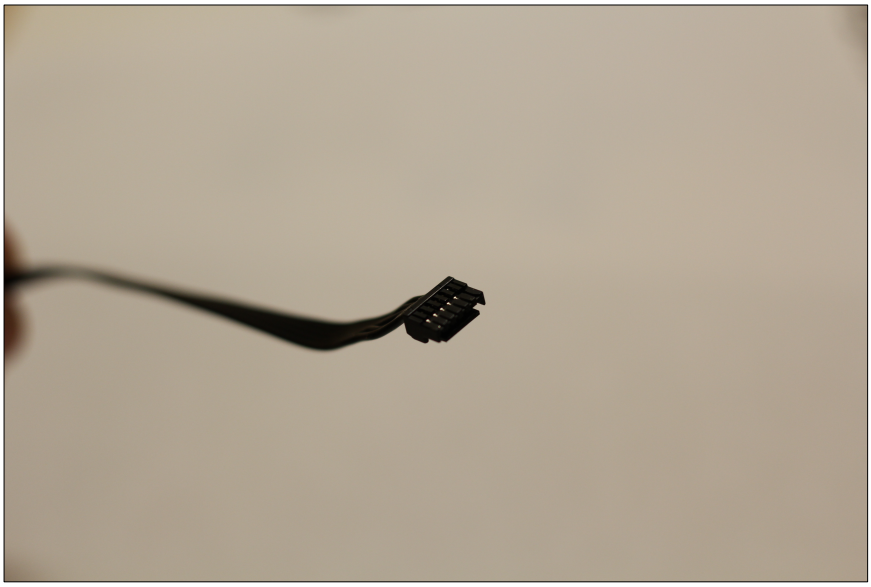


Figure 8: *Mōvi Pro COM to Mōvi Controller Receiver* Cable (Male Side)

[7] Female to Male DuPont wires, see **Figure 9**

[8] www.store.freeflysystems.com/collections/movi-pro/products/movi-pro-com-to-movi-controller-receiver-cable, see **Figure 8**

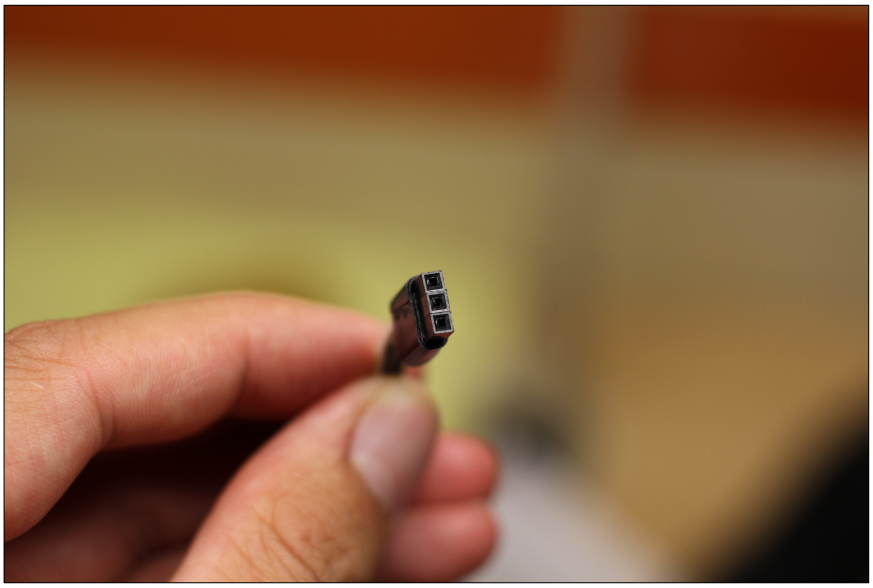


Figure 9: OUT-B Port F-M DuPont Wires (Female Side)

The wire successfully connects the Arduino to the MōVI.

5. Software Description

5.1 Arduino Software

To send commands from the Arduino Esplora to the MōVI, you must load Arduino code onto the Esplora. Code is written, processed, and uploaded to Arduino controller on the Arduino Web Editor. See **Figure 10** for the Arduino Web Editor layout.

Note: This report assumes the reader has a basic understanding of the Arduino[9] software configuration.

[9] Learn more at www.arduino.cc

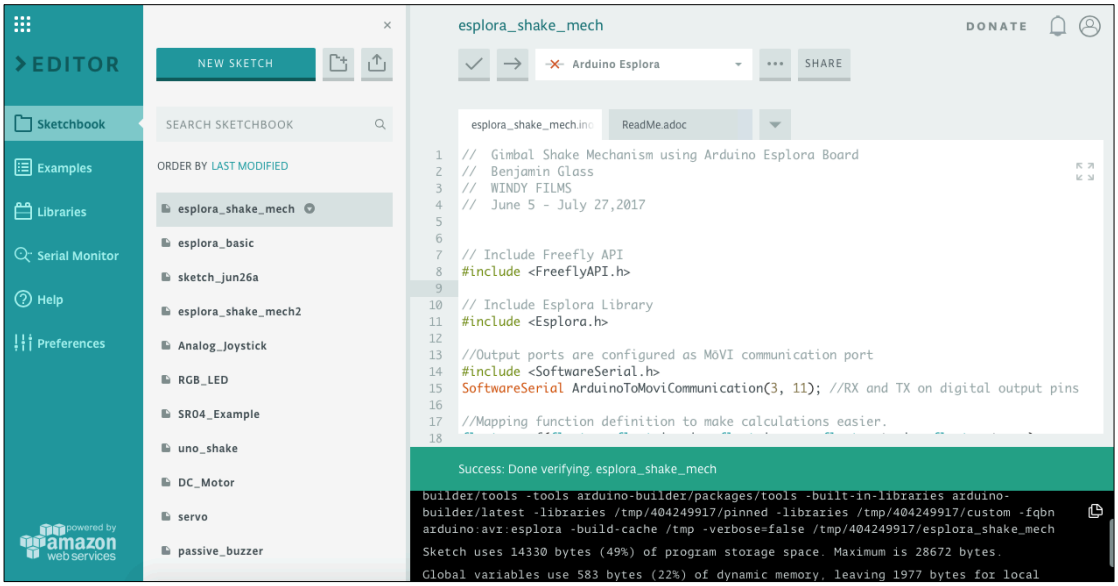


Figure 10: Arduino Web Editor

5.2 Freefly API

The Freefly API is what allows communication between the Arduino and the MōVI. The Freefly API comes as a set of files written in C and Objective-C. There was no need to update or change the given files in any way for the project.

To change the rotational velocity of the gimbal motors, we needed access to two specific MōVI commands: (1) to change the control type[10] and (2) value of pan and tilt.

```
FreeflyAPI.control.pan.type = ;
FreeflyAPI.control.pan.value = ;
```

Using the API allowed us to write these commands in the Arduino code. In order to use the API, the files needed to be uploaded as a library in the Arduino Web editor, and the library included in the code file (declaration in the beginning of the Arduino code file, line 8 in **Figure 10).**

5.3 Code Overview

For full Arduino code documentation, see the noted Github repository[11]. The Arduino code is written in the Arduino programing language.

[10] see "Freefly API Version 1.0." API structures and Gimbal and Lens Control Overview– control types include RATE (units are degrees/second), ABSOLUTE (units are the angle), and DEFER

[11] Github repository @glassb: shake-mechanism “www.github.com/glassb/shake-mechanism”

The algorithm is as follows:

Setup

1. Declare increment variables, `incr1` and `incr2`, respective to pan and tilt.
2. Set pan and tilt to control type RATE.

Loop

3. Read in the Esplora joystick in the X direction, the joystick in the Y direction, and the slider values. (Variable names in code: `joystickX, joystickY, sliderValue`)
4. Declare `shake_num_pan` and `shake_num_tilt` variables as sin functions, dependent on increment variables `incr1` and `incr2,` respectively. (As the increment values change, `shake_num_pan` and `shake_num_tilt` variables fluctuate between positive and negative values – see **Figure 11**)
5. Set the value of pan to the sum of `-joystickX` and `shake_num_pan`; likewise set the value of tilt to the sum of `joystickY` and `shake_num_tilt.`
6. Increment/reset `incr1` and `incr2.`

This manifests as the following:

- `shake_num_pan` and `shake_num_tilt` evaluate to small values that are added to the value of pan and tilt respectively. The GCU adjusts for the addition of the added value by physically correcting pan and tilt rates.
- The increment value changes at the end of the algorithm, and the Arduino loop resets, causing `shake_num_pan` and `shake_num_tilt` to change value at each iteration of the loop.
- This creates constantly changing motion in pan and tilt.

The following is an abridged version of the algorithm, and gives a syntactical example of the code:

```
(0) float incr1 = 1;

(1) float shake_num_pan = sin((incr1/60)*2*pi)/2;

(2) FreeflyAPI.control.pan.value = -(joystickX) + shake_num_pan;

(3) incr1 += 1;
```

In line (0), we declare the increment variable `incr1`, which increases by 1 with each loop of the Arduino code. In line (1), `shake_num_pan` is declared as a variable, a sin function, and value dependent on `incr1`. Because the variable `incr1` is changing at every loop of the Arduino code,

`shake_num_pan` is changing as well. In line (2), the MōVI's pan value is equal to the -`joystickX`[12] value summed with `shake_num_pan.` This extra `shake_num_pan` gives a constant jitter in the background, while the joystick can still control pan. In line (3), `incr1` is updated for the next iteration of the loop.

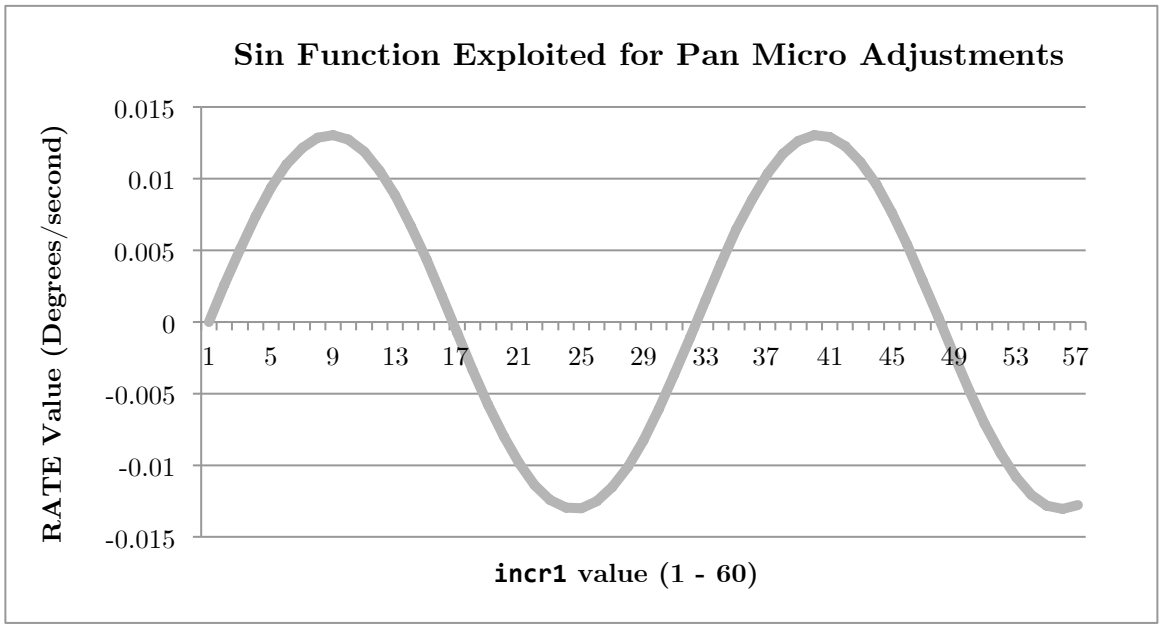


Figure 11: Graphical interpretation of pan motion

Figure 11 illustrates the micro adjustments that result from the added value of `shake_num_pan` over a series of increment changes. Because the speed of the Arduino loop can be dictated by the Arduino delay function, I have the capability to control how fast pan and tilt change values. Furthermore, the period and amplitude of the sin function can be changed to add more or less joggle to the frame. The manipulative properties of a sinusoidal function mean that **Figure 11** can take many different forms, with peaks and troughs that are higher or lower, as well as wider or skinnier. In general, the function period is under one second.

6. Limitations

The mechanism is functional, yet many hardware problems inhibit the operator from utilizing the mechanism in its intended way.

[12] We negate `joystickX` for physical continuity.

1. When I first started testing the Arduino board with the MōVI Pro, the camera would slowly veer to the right when I wasn't touching the joystick. This was because the Arduino Joystick was not perfectly centered, and was tilted slightly to the southwest direction at all times. Even though it was only slightly off center, any direction in the joystick triggers pan and tilt to move. Maybe this was due to the cheapness of the board. I had to fix this problem in code. Because the joystick was off center by only a tiny bit, I set all values less than .04 from the Joystick to zero. This stopped the drifting. In general, this points to the lack of precision in the Arduino controller.

2. The cable that connects the Arduino to the MōVI is about two feet long. Because the GCU moves with the camera, the cable sometimes goes taught when the camera rotates far enough. The cable inhibits camera movement, and needs to be attached to the MIMIC, in which case, the MōVI would again be wirelessly controlled.

3. There are durability issues with the Arduino. The company makes no official case for the board, so the Esplora is exposed to breakage. The slider is also fragile, and flops over ever so slightly. For an instrument that will be brought into the field and be physically tested, the controller and wire combination need to be sturdy enough to withstand drops, lightweight, and light misuse. Otherwise, I could see the Arduino board cracking and its parts malfunctioning.

References

Freefly Systems. "Freefly API Version 1.0." *Google Docs*. Google, 6 Mar. 2017. Web. 10 July 2017. <https://docs.google.com/document/d/1eNkpmxHHc22ooSi0EKxd6F0UNi5H_kI13hzm_rTR8i0/edit>.

Freefly Systems. "Freefly API for Makers - Intro." *Google Docs*. Google, n.d. Web. 10 July 2017. <https://docs.google.com/document/d/16L65isO7Ifh3iWyqnqK69DHKzTWpJYlfl-hDMx7ABkY/edit#>.

Freefly Systems. "Tutorial 02 - Classic Gamepad." *Google Docs*. Google, n.d. Web. 10 July 2017. <https://docs.google.com/document/d/10XTM--pDWMwRonMOx_Hc7SlfVGNe0F_PXkrJu7DgCc4/edit#heading=h.dz4k5uhvpwvm>.

Learn about the Upcoming MōVI Pro API. FB LIVE 3/2/2017. Freefly Systems, 2 Mar. 2017. Web. 5 June 2017. <https://www.youtube.com/watch?v=UsawsEL9t9E>.

MōVI XL Q&A, API Programming, and Shots Executed Using Bush Pilot. FB Live 3/23/17. Freefly Systems, 23 Mar. 2017. Web. 5 June 2017. <https://www.youtube.com/watch?v=Lvy7YOxZF7w&t=2085s>.

Barela, Mike. "The 21st Century Digital Home." *Arduino Esplora Tinkerkit Outputs*. N.p., 01 Jan. 1970. Web. 10 July 2017. <http://21stdigitalhome.blogspot.com/2013/01/arduino-esplora-tinkerkit-outputs.html>.

Acknowledgements

I would like to thank Harvey Burrell for his willingness to supervise this term project. The rest of this Windy Films team should also be acknowledged; thank you to Will Humphrey and Tripp Clemens for their support while working in Studio 16.

I would like to thank Talamas Company for supplying the MōVI Pro for testing.

As well, thank you to Freefly Systems for quick and meaningful responses to my questions, and for their multitude of online resources.