Artificial Shake Mechanism -
For Freefly MōVI Pro by Arduino Microcontroller

Windy Films Internship Project Report

Benjamin Glass

July 27, 2017

Abstract

With camera movement, cinematographers seek for 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 flux of a car moving fast over bumpy terrain, or the slow weaves and jostles that are characteristic of a camera handheld, these frame motions needs to be deliberate. With machinery, we have the ability to pinpoint the frame motion we desire, eliminating the human element – and likely the associated error.

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

This mechanism is functionally effective, although lacks usability and flexibility in some areas due to the fragility of the Arduino controller and inaccessibility of the Arduino code to the cinematographer. In the development process, I learned about basic electronics, the Arduino microcontroller family, the Arduino web editor, the MōVI Pro mechanics, and the Freefly API.

Introduction

Timeline and Assignment

From June 5, 2017 until July 27, 2017, I joined the team at Windy Films[1] as an intern, under the supervision of Harvey Burrell.

My term assignment 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 of the 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 implant a sense of stir in the viewer. This mood would not as nearly be conveyed if the camera were steady.

It was important to not stray too far away from this basic 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.

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.).

Personal Objectives included:

- Understanding basic electronic systems, the Arduino family, the Freefly Application Programing Interface, and the MōVI Pro mechanics
- Writing well-documented and efficient code
- Advance ability to communicate problems and solutions
- Learning how to research new subjects, and where to find answers to questions

[1] Located in Boston, Massachusetts. windyfilms.com

I had no experience with electronic hardware before this project. I had been intensively programming and writing code for almost two years at the start of June 2017.

Methods

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.

MōVI Pro[2]

The MōVI Pro is a *camera movement system* made by *Freefly Systems*. The MōVI was optimal for this project because of its reputation as a professional filmmaking tool, its ability to securely handle a camera, and its accessible API[3].

Image 1 – 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.

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

[3] Application Programming Interface: Wikipedia states as, "...a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer."

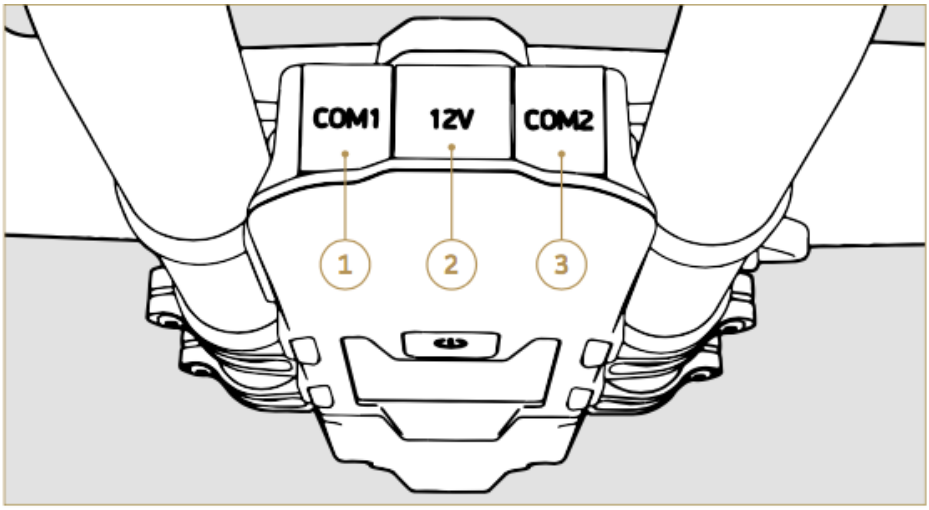


Image 2 – Gimbal Control Unit with Numbered Ports (Top-view)[4]

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 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 controls.

Although the official MōVI controller[5] is tailored directly for MōVI Pro use, Freefly Systems allows more capabilities. In March of 2017, Freefly publically released their API[6], 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.

Arduino Esplora Hardware Controller

The Arduino[7] microcontroller platform suited my needs for the custom controller. 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. Freefly has also documented and published Esplora-MōVI connections using the Freefly

[4] MōVI Pro Operation Manual, page 21
[5] See "MōVI Controller" on Freefly System's Website
[6] In this case, the commands used to control the MōVI Pro were now public.
[7] "Arduino...an open-source electronics platform based on easy-to-use hardware and software."

API[8]. By following their documentation[9], I was able to setup an API connection and basic code structure for my project.

The Esplora controller runs for around $43 on the Arduino website.

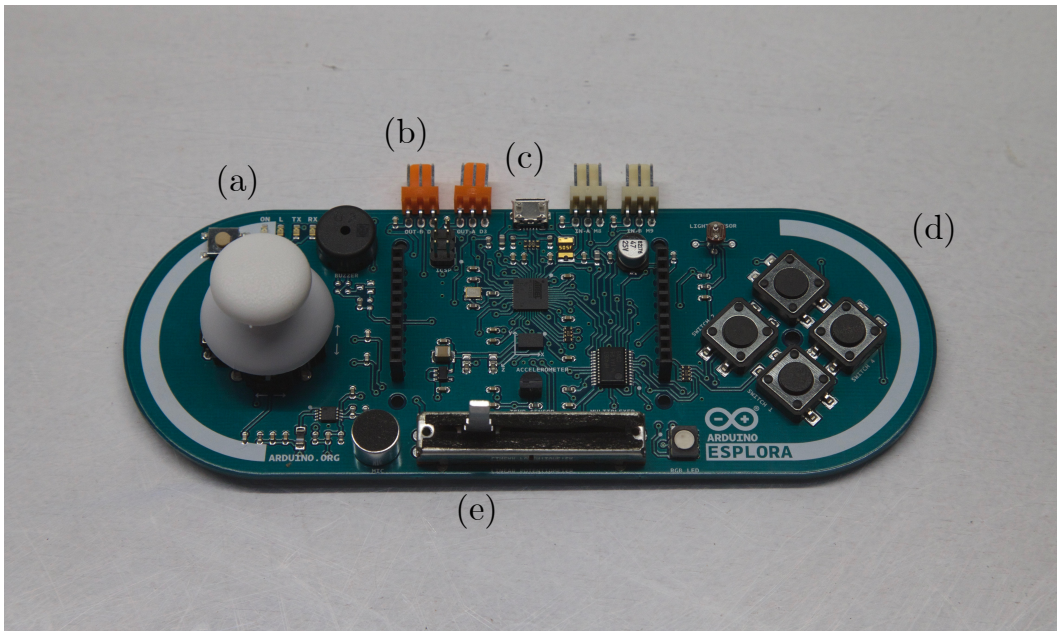


Image 3 – Arduino Esplora Microcontroller
- **(a)** – Arduino Joystick
- **(b)** – 3-Pin OUT-B Port (see Table 1)
- **(c)** – USB port (power)
- **(d)** – Arduino buttons (4)
- **(e)** – Arduino slider

[8] See References - "Tutorial 02 - Classic Gamepad."

[9] Freefly released a *Freefly API for Makers* document that explained how to setup an Arduino Esplora to control a MōVI Pro's basic mechanics. It explained both the hardware and software components. From this well-documented manual available from Freefly, 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 Arduino code that comes from Freefly's documentation includes lines 8-21, 30, 34, 57, 58, 61, 134-195.

Arduino-MōVI Pro Cable

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

The Arduino has a three-pin OUT-B port, which provides access as follows:

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

Table 1 – OUT-B Port Pin Diagram

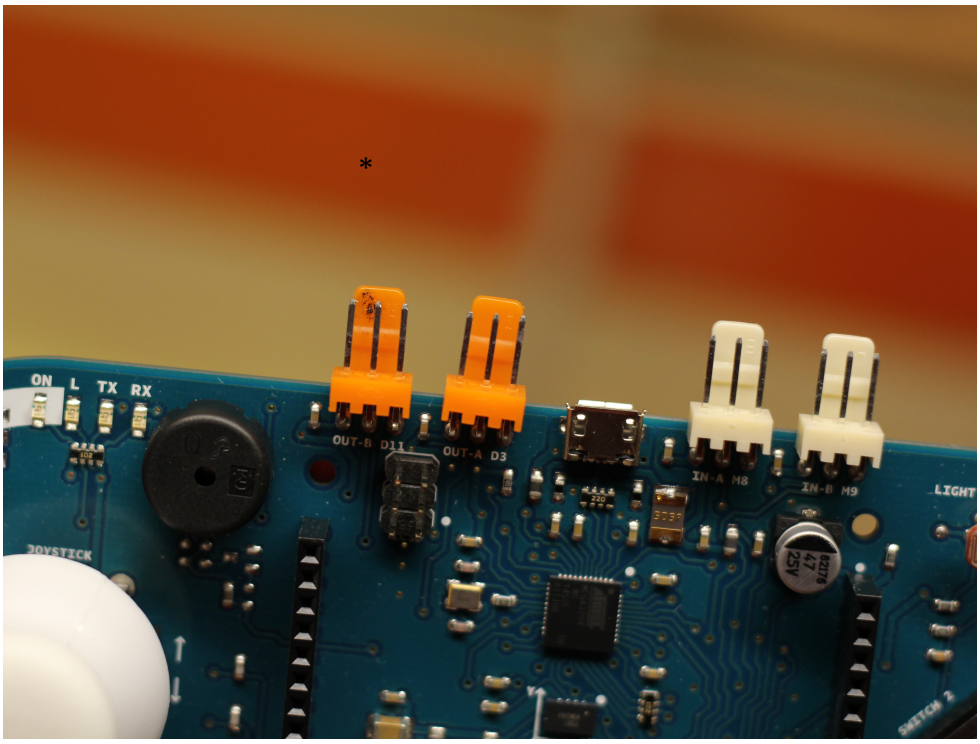


Image 4 – OUT-B Port

The MōVI Pro COM1 Port is on the left side of the GCU (see Image 2). It is of type JST GH 6-PIN, as follows:

[10]

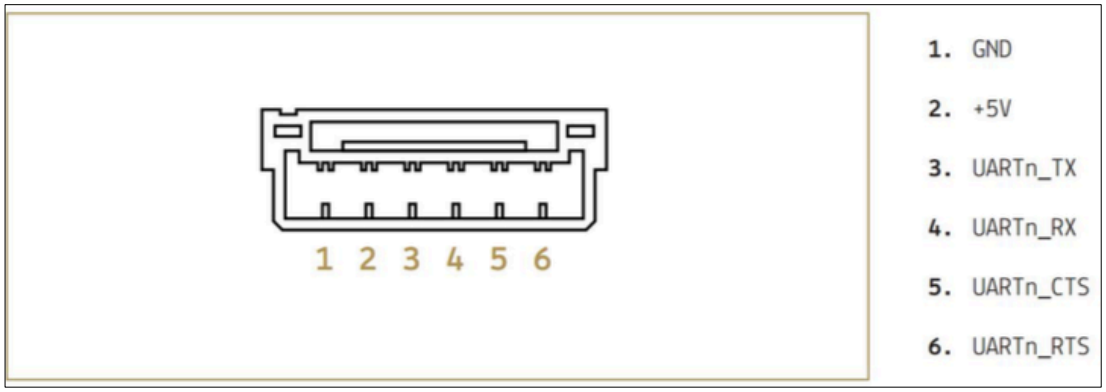


Image 5 – 6-pin diagram

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) connects to UARTn_RX in COM1. The schematic is as follows:

Esplora-MōVI Port Connection		
***Esplora* OUT-B**		***MōVI Pro* COM1**
GND	-	GND
+5V	-	+5V
		UARTn_TX
Digital Pin 11 (TX)	-	UARTn_RX
		UARTn_CTS
		UARTn_RTS

Table 2 – Arduino to MōVI Schematic Table

[10] MōVI Pro Operation Manual, page 23

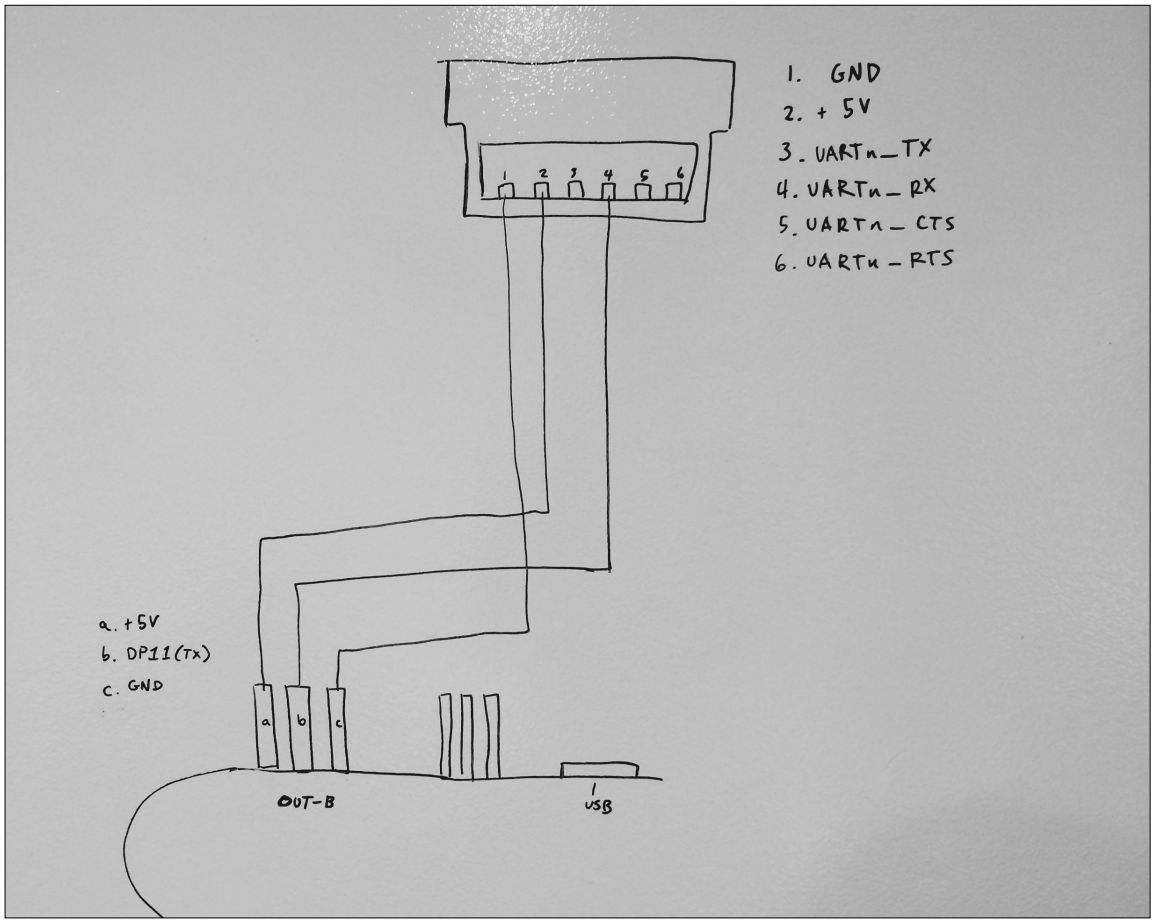


Image 6 – Arduino to MōVI Schematic

Table 2 and Image 6 illustrate the Esplora-MōVI schematic.

The Arduino-MōVI cable was comprised of three separate segments of wires. I used three F-M wires[11] from a basic Elegoo Electronics kit to connect to the 3-pin OUT-B port on the Arduino. I wrapped the three female connectors with electrical tape.

I used a *MōVI Pro COM to MōVI Controller Receiver* Cable[12] 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).

[11] Female to Male DuPont wires, see Image 8
[12] Cable on Freefly's Website, see Image 7

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, connecting them all with electrical tape. I then wrapped the three unused MōVI cable wires with electrical tape to tidy the whole cable.

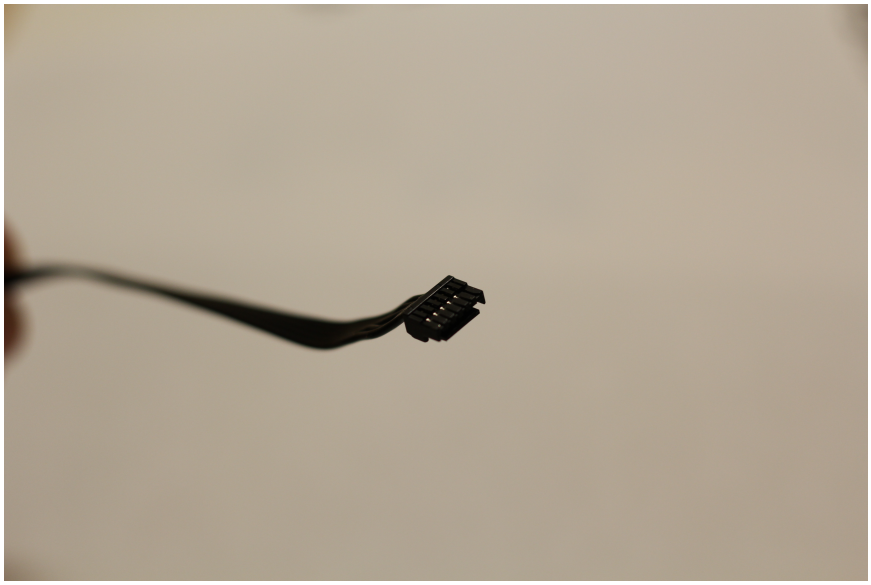


Image 7 - *MōVI Pro COM to MōVI Controller Receiver* Cable Male Port

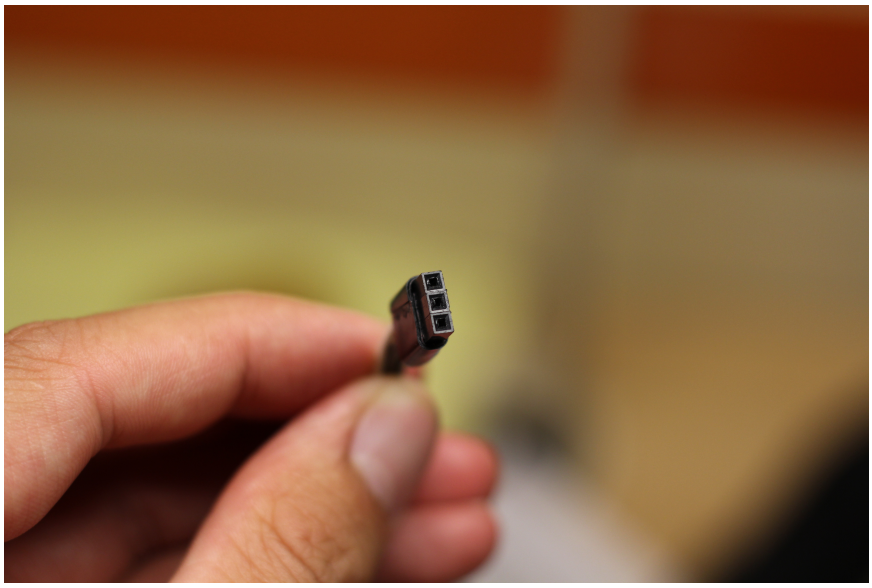


Image 8 – OUT-B Port F-M DuPont Wires (Female Side)

The wire successfully connects the Arduino to the MōVI.

Note: The Arduino controller must be attached to a power source, either through an external battery or laptop via USB connection.

Arduino Software and Freefly API

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.

The Freefly API[13] is what allows communication between the Arduino and the MōVI. By including the API (declaration in beginning Arduino code file), we have the ability to write the MōVI commands in the Arduino text file.

Code Overview

For full Arduino code documentation, see the noted Github repository[14]. In short, the Arduino microcontroller controls gimbal motors via the GCU, constantly changing their rotational speed and direction to allow for jerky pan and tilt camera movement.

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 **Graph 1**)

[13] See reference section for full Freefly API documentation - "Freefly API Version 1.0."
[14] Github repository @glassb: shake-mechanism "www.github.com/glassb/shake-mechanism"

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:

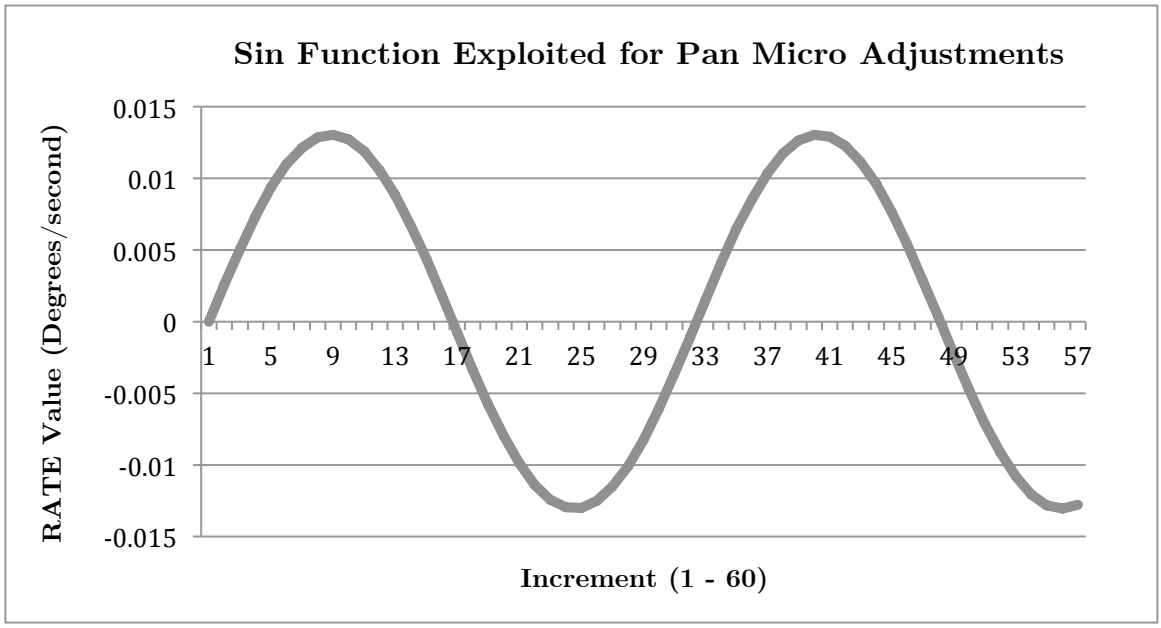
- `shake_num_pan` and `shake_num_tilt` evaluate to small values that are added to the value of pan and tilt respectively. The GCU adjusts for the addition of the added value by correcting pan and tilt.
- The increment value changes at the end of the algorithm, and the Arduino loop resets, causing `shake_num_pan` 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)/8;

(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, and resets to 1 when it equals 60. 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`[15] 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.

[15] we negate `joystickX` for physical continuity



Graph 1

Graph 1 illustrates the micro adjustments that come from the result of `shake_num_pan` over a series of 60 increment changes (60 loops in the Arduino code). 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. In general, the function period is under one second.

The basic motion is quite repetitive, unlike real camera shake, that is random and fluctuates. I have a variable that controls the amplitude of the sin function. This variable is set at the beginning of every Arduino loop to a random integer value between 1 and 5. This creates spontaneous surges in camera movement.

This shake mechanism runs constantly in the background of the joystick controller, which can be freely operated as well to adjust pan and tilt. The mechanism suits a two-person operation team, one person holding the gimbal, and the other controller direction of the camera while the Arduino automatically adds shake to the frame.

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>.

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.

Thank you to my parents and my sister for all their support too.