AN EXTENSIBLE SYSTEM FOR PHYSICALLY-BASED VIRTUAL CAMERA CONTROL USING ROTATIONAL MOTION CAPTURE

A Thesis

by

ROBERT SHELBY HUEBEL

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Philip Galanter
Committee Members,	Tim McLaughlin
	John Keyser
Head of Department,	Tim McLaughlin

May 2014

Major Subject: Visualization

ABSTRACT

An important characteristic of any well-designed system is user interface design. Even in extremely complex systems with high learning curves, a designer's goal is to create an interface that is simple to use and that feels natural, while at the same time allowing the user full access to all of the system's functionality.

In the field of visual effects and computer animation, effective manipulation of the virtual camera is essential to any successful production. In most cases, the virtual camera is manipulated with standard input devices such as the mouse and keyboard. This thesis presents an alternative user interface design for virtual cameras using physical controls.

This thesis describes an extensible system which offers physically-based, real-time virtual camera control in a commercial 3D software package by means of rotational motion capture. The system described is composed of both programmed physical hardware and a software application. The physical hardware consists of multiple analog sensors programmed to work in conjunction to detect rotation about the device's three axes. The software application interprets the data sent from the physical hardware into a form a 3D software package can use to manipulate a virtual camera, while also providing additional functionality for the motion capture and camera manipulation processes.

This thesis constructs the physical motion-sensing device using only affordable, commercially-available parts. The software components of the system use freely available programming languages and development environments. The system possesses the ability to be expanded upon to add additional functionality or modify existing components, in order for it to be flexible enough for numerous applications.

The result of this research is a working prototype system that captures pitch, roll, and yaw motions that mimic actual physical motions performed by the user. The workflow provided by the system allows for quick iterations and facilitates an interactive control scheme for the virtual camera that traditional mouse-and-keyboard techniques cannot.

DEDICATION

To my wife Adrienne, who continues to inspire, support, and encourage me in everything that I do. Thank you for loving this goofy nerd, even during the rough times. I can't wait to see what life holds for us.

To my parents Robert and Dorothy Huebel, who are the absolute best parents a guy could have asked for. Your constant love and support has made me the person I am today, and for that I am eternally grateful. I hope I have made you proud.

To everyone that has shown me kindness throughout the years, no matter how small or insignificant. It did not go unnoticed, and every bit was appreciated.

ACKNOWLEDGEMENTS

I would like to thank professor Philip Galanter, my committee chair, for all the support and guidance he has offered during the many, many months it took to bring this thesis to completion. I am especially grateful for his patience and understanding, both of which were absolutely essential for me to get to the end of this. I would like to thank my other committee members, professor Tim McLaughlin and professor John Keyser, for sticking with me throughout the whole process and always responding to my numerous but sporadic emails.

I would like to thank all the students of the Vizlab for making it such a creative, inspiring, and educational experience. I would especially like to thank Cassie Hanks, without whom I never would have managed to get all my OGS forms signed, and my "Pasadena Trio" of former Vizzers, namely Jose Guinea Montalvo, Megha Davalath, and Adan Pena, who answered any and all questions I posed to them about writing this paper and provided invaluable moral support as I worked to finish it.

Thanks also to all the faculty and staff of the Department of Visualization for all your time and help during my years there.

Finally, a special thank you goes to my family, especially my wife Adrienne and my parents Robert and Dorothy, for never giving up on me and never letting me give up on myself.

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGEMENTS v

TABLE OF CONTENTS vi

LIST OF FIGURES viii

1. INTRODUCTION 1

2. PRIOR WORK 6

2.1 Historical Virtual Camera Development 6
2.1.1 Image Capture 7
2.1.2 Camera Motion 7
2.1.3 Manipulation 8
2.2 Recent, Production Specific Virtual Camera Development 9
2.2.1 Surf's Up (2007) 9
2.2.2 Wall-E (2008) 11
2.2.3 The Polar Express (2004) 12
2.2.4 Avatar (2009) 14
2.3 Motion-Sensing Systems 14
2.3.1 Accelerometers 15
2.3.2 Gyroscopes 19
2.3.3 Magnetometers 22

3. METHODOLOGY 24

3.1 User Experience 25
3.2 Data Production Method 26
3.2.1 Hardware Selection 27
3.2.2 Sensor Initialization 29
3.2.3 Data Interpretation 31
3.2.4 Data Output 36
3.3 Artistic Control 38
3.3.1 Physical Device 38
3.3.2 Extensibility 39
3.3.3 Software Application 39

4. IMPLEMENTATION 41
4.1 Data Production Method 41
4.1.1 Hardware Selection 41
4.1.2 Sensor Initialization 45
4.1.3 Data Interpretation 50
4.1.4 Data Output 61
4.2 Artistic Control 62
4.2.1 Controls 63
4.2.2 Workflow 69
5. RESULTS 71
5.1 Overview 71
5.2 Tests 71
5.3 Strengths and Weaknesses 75
6. CONCLUSION 78
7. FUTURE WORK 80
REFERENCES 82

LIST OF FIGURES

FIGURE Page

1.1 The Nintendo Wii Remote. [19] 2

1.2 A still frame from the 2007 Sony Pictures Animation film *Surf's Up* in which the character Cody Maverick addresses the camera. [21] . . 3

2.1 An example of Kolb's results. Each frame is shot from the same position, but simulates a different real-world lens configuration. [30] . 7

2.2 (left) James Williams holds the camera control rig used on Surf's Up. [33] (right) A frame from *Surf's Up*. [21] 9

2.3 Cardboard stand-ins of characters Wall-E and Eve were used to perform lens tests for the movie *Wall-E*. [36] 11

2.4 Robert Zemeckis works with a virtual camera operator on the virtual set of *The Polar Express*. [23] 12

2.5 James Cameron uses one of the virtual camera control rigs on the set of *Avatar*. [34] 13

2.6 A hypothetical accelerometer being static in a weightless environment. [8] 15

2.7 A hypothetical accelerometer that has been moved quickly to the left in a weightless environment. [8] 16

2.8 A hypothetical accelerometer that is sitting on the ground and experiencing Earth's gravity, but is otherwise static. [8] 17

2.9 A hypothetical accelerometer that is sitting on the ground, is experiencing Earth's gravity, and is tilted 45 degrees clockwise around the Y axis. [8] 18

2.10 A visual example of the Coriolis effect at work. [13] 20

2.11 A Foucault pendulum. [42] 21

2.12 A standard handheld compass. [16] 22

3.1 The proposed setup for the system. The physical device is represented by a red pack of cards. Autodesk Maya is open in the top left of the screen, and a mockup of the software application is open in the bottom right of the screen. 25

3.2 One version of Arduino, the "Duemilanove" board. [11] 27

3.3 Part of a typical specification sheet for an accelerometer. [7] 30

3.4 An analog trigger control. [4] 39

4.1 The Sparkfun Electronics SEN-09623 9DOF Razor IMU. [6] 42

4.2 The physical device, pictured with the top of the plastic enclosure removed. 44

4.3 A diagram showing the connections between the different components. 45

4.4 The software application. 62

4.5 The software application showing the connection help screen. 63

4.6 The software application showing the hotkeys interface. 64

4.7 The software application in the midst of recording keyframes after the record button has been pressed. 65

4.8 The axis selection control, showing x and z axis as active and y axis as inactive. 66

5.1 A screenshot from the video of the first test. 72

5.2 The setup used to compare motion captured by the system in this thesis to real-life camera motion. 73

5.3 A series of screenshots from a test video comparing roll captured with the system from this thesis (top row) to real-life camera motion (bottom row). 74

5.4 The same two sets of screenshots from Figure 5.3 are now superimposed on top of one another at reduced opacity. 75

1. INTRODUCTION

Cinéma-vérité is a style of documentary filmmaking where the camera action has an improvised and interactive feel. Sometimes it is used during action scenes to make them seem more gritty, realistic, or visceral. Some movies and TV shows are filmed entirely in a faux-documentary style. They use hand-held camera movements to make shots seem unscripted and more impromptu, like a camera crew was really present at the time of filming. Regardless of how it is used, cinéma-vérité and the hand-held camera look rely heavily on the physicality of the cinematographer or camera person. While the general blocking of the camera can be planned ahead of time, much of the final look is determined "in the moment." A camera operator's instincts, on-the-fly decision making, reflexes, and physical limitations all contribute to the visual look of the final shots.

In the field of visual effects and computer animation, the look and feel of hand-held camera action can be reproduced in a virtual environment by talented technical artists using standard input devices such as the mouse, tablet, and keyboard. However, this process lacks the physicality of camera use and is highly abstracted from the goal and significant time and effort are required. If done improperly or poorly, the resulting scene can appear either too mechanical or too wildly active, failing to invoke the natural compositions as if there had been a human behind the lens. In addition, the high level of abstraction hinders important creative members of the crew, such as the director or cinematographer, from quickly taking control of the virtual camera to test out a new angle or camera motion in the scene.

One approach to tackling this problem, taken by several professional production studios, is to employ advanced motion capture technology to track the motion of a

physical camera, or a similar device, operated by a real person in a stage environment. The resulting motion data is applied to a virtual camera with varying degrees of real-time playback. While this approach has become more commonplace in recent years, the system required is only viable for large professional studios that can afford to provide the specialized environment, equipment, and crew to run and maintain it.

The goal of this research is the development of a system for creating physically-based camera motion that can be both feasibly implemented and easily expanded upon by a small studio or an individual. The ability for the system to be expanded upon is particularly important, as the needs and requirements of such a system can greatly vary based on both the project or production it is being used for and the individual using it.

Figure 1.1: The Nintendo Wii Remote. [19]

The primary technical inspiration for this thesis is the Nintendo Wii Remote, pictured in Figure 1.1. Often referred to as the Wiimote, the Nintendo Wii Remote is the primary controller for Nintendo's Wii console, which was released in November 2006 [18]. A primary feature of the Wii Remote is its motion sensing capability. It utilizes an accelerometer to detect physical gestures and an infa-red camera to allow

pointing [19]. The system allows users to interact with and manipulate items on screen by means of these physical gestures and pointing. The remote, in conjunction with the Wii console, was one of the earliest examples of motion sensing technology that was available to the general public. For the first time, a virtual bowling ball could be rolled down the lane by mimicking the actual motions involved with bowling. A sword could be swung not by pressing a button, but by making a swiping motion in front of one's body. The intuitive nature of the controls made the Nintendo Wii extremely popular with "non-traditional" and first-time gamers [18].

Figure 1.2: A still frame from the 2007 Sony Pictures Animation film *Surf's Up* in which the character Cody Maverick addresses the camera. [21]

The primary visual inspiration for this thesis is the 2007 Sony Pictures Animation film *Surf's Up*. The film follows a young rockhopper penguin named Cody Maverick on his quest to succeed in an upcoming surfing competition. Taking inspiration from such surfing documentaries as *The Endless Summer* and *Riding Giants*, the filmmakers wanted the movie to feel like a documentary, even though it was animated

[38]. Several different measures were taken in order to achieve the desired look and feel. The shots in *Surf's Up* are much longer than a typical animated film, and the dialogue has more of an improvised feel. In addition, throughout the film the animated characters acknowledge the presence of a camera crew, either by talking directly to the camera during an interview, as seen in Figure 1.2, or glancing quickly at the camera when they are nervous about being filmed. The fake camera crew can even be heard audibly talking in a few scenes. Arguably the most significant aspect to contribute to the documentary feel was the camera work. The layout department developed a sophisticated rig for the virtual camera that allowed them to add a realistic physicality to their camera movements by moving around with a real camera in a real space and capturing the motion [33]. This resulted in a believable hand-held look for the film, setting it apart from other animated films at the time and winning over critics and audiences with its authentic approach and feel.

The system proposed in this thesis is composed of three components: a physical device, microcontroller code for the physical device, and a software application. The physical device must contain a sufficient number and variety of both sensors and other electronic components in order to accurately detect rotation in all three dimensional axes. Being well-constructed is also important because the device is intended to be picked up and moved around, and the user should not have to worry about the device breaking or falling apart while being handled. The physical device is nearly useless without its programming code, which interprets the signals sent by the various sensors of the device and converts them into a form that can be used for motion sensing purposes. The programming code must be simple enough to fit in a limited amount of memory available on the device. Efficiency is also essential, so that running the necessary calculations and outputting the results does not introduce a noticeable amount of lag to the real-time system. The software application serves

as a link between the output of the physical device's programming code and the 3D software package that provides the virtual camera and environment. It connects to the 3D software package and both sends and receives data from it. The application provides a graphical user interface that allows the user to quickly perform all the necessary functions required to use the system, while also adding additional useful functionality overall.

As part of an animation production pipeline, this system will be used by the layout department when authentic-feeling camera motion is needed. Rather than tediously creating keyframed animation curves by hand that lack realism, the user can perform the actual rotational movements they want their virtual camera to have by using this system. This will result in quicker iterations, and will allow for interactive art direction from a director or cinematographer. The anticipated impact is a quicker and more efficient workflow that produces more realistic looking rotational camera motion than traditional techniques.

This thesis describes a prototype system, provides visual examples of the possible uses of the system, and describes the limitations of the system. In the future work section, potential improvements that can be made to the system are discussed, and various applications for which the system could be adapted for use beyond its original scope are theorized.

2. PRIOR WORK

Previous research includes examining both the historical development of the current standard for virtual cameras as well as more recent, specific efforts at expanding the usability and functionality of the virtual camera. In addition, we examine several of the currently available real-world physical sensors, focusing on how they operate, what kind of data they can provide, and what their limitations are.

2.1 Historical Virtual Camera Development

Camera control has always been an integral part of 3D software packages. In the early years of computer graphics, researchers developed what would now be considered basic or standard functionality for virtual cameras. It is important to note that there is not a direct correlation between what a real camera does well and what a virtual camera does well. They are two completely separate systems with different requirements and restrictions. As a result, the initial implementations of virtual cameras did not always mimic how real-life cameras worked. However, as time progressed and technology advanced, computer graphics professionals worked to give virtual cameras similar functionality to real cameras.

2.1.1 Image Capture

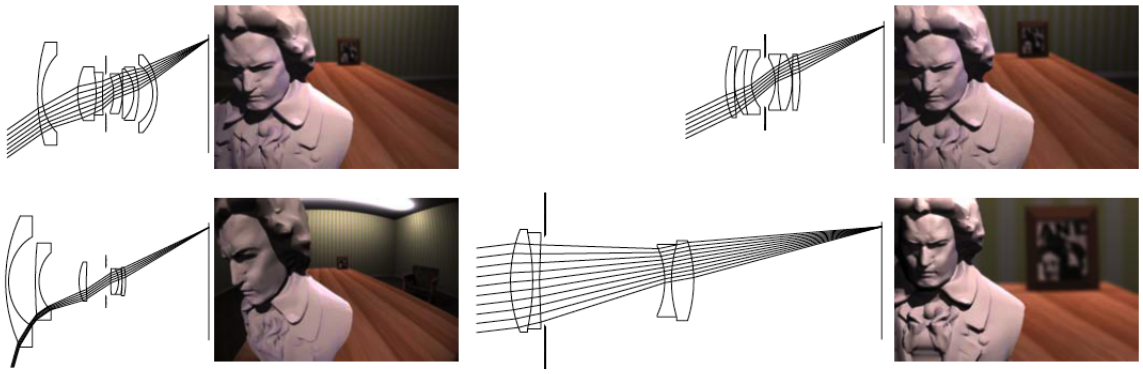

Figure 2.1: An example of Kolb's results. Each frame is shot from the same position, but simulates a different real-world lens configuration. [30]

Initially most rendering algorithms were limited to using pin-hole camera projection geometry which was developed for display of 3D line drawings on vector devices. Potmesil *et al.* developed a more complex algorithm designed specifically for raster displays that simulated the effects of a lens and aperture on a real camera. This expanded the functionality of the virtual camera, allowing for cinematic techniques such as uniform defocusing of a scene, depth of field, lens distortions, and motion blur [31]. Kolb *et al.* later developed a virtual camera model that incorporated a virtual lens system and film backplane [30]. The lens system consisted of a sequence of simple lens elements, stops and apertures. The camera simulation module would compute the irradiance on the film backplane from the scene radiances using distributed ray tracing. This allowed accurate simulation of the behavior of real-world physical cameras and lens systems.

2.1.2 Camera Motion

To animate the camera and allow it to move in 3D space, a technique called keyframing was carried over from traditional animation. Using this technique, an

animator or layout artist sets discrete positions and orientations for the camera at important, or "key", frames in shot. Once several keyframes are set, the 3D software interpolates the position and orientation of the camera for the all frames in between the keyframes. Initially only linear interpolation between keyframes was possible. Kochanek *et al.* developed a method that allowed animators to manipulate the tension, continuity, and bias of the splines generated during keyframing, providing more control over frame interpolation and thus smoother, more naturalistic motion [29]. Shoemake's work with quaternion curves greatly improved the interpolation of rotations. By using the four coordinate system to represent rotations instead of the traditional three coordinate system, Shoemake was able to eliminate earlier method's potential for quirky behavior such as gimbal lock, a condition where two of the three axes of rotation are driven into a parallel configuration, "locking" the system into rotation in a degenerate two dimensional space [35].

2.1.3 Manipulation

Using 2D devices such as the mouse and keyboard to operate a camera with 6 degrees of freedom can be difficult, so researchers developed camera control schemes based on easily understood metaphors. These metaphors helped the user to initially understand the system's behavior given different types of input actions. A good metaphor was both appropriate and easy to learn. Some examples of these metaphors, outlined in Ware *et al.*, include the "eyeball in hand", the "scene in hand", and the "flying vehicle control" [40]. Each metaphor has its strengths and weaknesses, and some continue to be used as control schemes today. As technology progressed, 3D input devices such as the Spaceball, Polhemus and DataGlove were developed, providing additional input methods for virtual camera control [39].

2.2 Recent, Production Specific Virtual Camera Development

In the animation and visual effects industry, studios must sometimes create their own solutions to camera control challenges that arise during production. The standard tools that are available may not be sufficient to achieve the visual look the director or cinematographer wants, or the standard workflow for producing virtual camera motion may be inefficient for the needs of the production. In this section, we examine some professional productions that expanded the functionality of their virtual camera systems to meet specific production requirements.

2.2.1 Surf's Up (2007)

For the 2007 film *Surf's Up*, Sony Pictures Animation wanted to create an animated documentary-style film about a group of penguin surfers. The protagonist, a teenage rockhopper penguin named Cody Maverick, leaves his home in Shiverpool, Antarctica, to compete in a surfing competition on the tropical Pen Gu Island. A film crew follows Cody's journey from start to finish, interviewing Cody and his companions and filming the surfing competition. The film that audiences see in the theatre is supposed to be the documentary created by this film crew.

Figure 2.2: (left) James Williams holds the camera control rig used on Surf's Up. [33] (right) A frame from *Surf's Up*. [21]

Achieving the documentary feel was key to the success of the film. According

to co-directors Buck and Ash Brannon, the story artists "boarded the film like any other animation, but the trick after that was to create the illusion of spontaneity. That you aren't quite sure what a character was going to do at any moment and you had one chance to catch it with a handheld camera" [33]. James Williams, the layout supervisor on *Surf's Up*, was tasked with developing a method to achieve the naturalistic look of a documentary film using a virtual camera. To do this, the layout team constructed a single-point mocap system composed of a Thirdtech HiBall tracking system and an old Sony analog video camera. The model of camera was chosen specifically for its weight; Williams wanted something that gave the operators the feeling of a real camera on their shoulders while they shot the action. The HiBall tracking system attached to the camera would look at an array of 3000 infa-red LEDs attached to the ceiling of a specialized studio in order to determine where it was in space. Additional sensors placed on the camera gave the operator the ability to replicate camera zoom while shooting and view a rough version of the animation through the video camera's viewfinder [33].

The system created by the layout department allowed the camera operator to follow the animated characters and interact with them in a "real" way, just as a real-world documentary filmmaker would. Because the operator of the system could see the action of the virtual scene in his viewfinder, he could respond to the performances of the animated characters in real-time. In addition to character shots, the system could also be used for other camera techniques. For example, to simulate the look of a camera filming from the back of a jet ski, the crew translated a camera in their 3D software package and applied camera shake from the real, motion-capture camera. For a virtual helicopter shot, they used motion paths to describe the motion of a virtual helicopter in the environment and then applied rotations from the motion-capture camera to simulate the feeling of a camera crew looking out the window

[33].

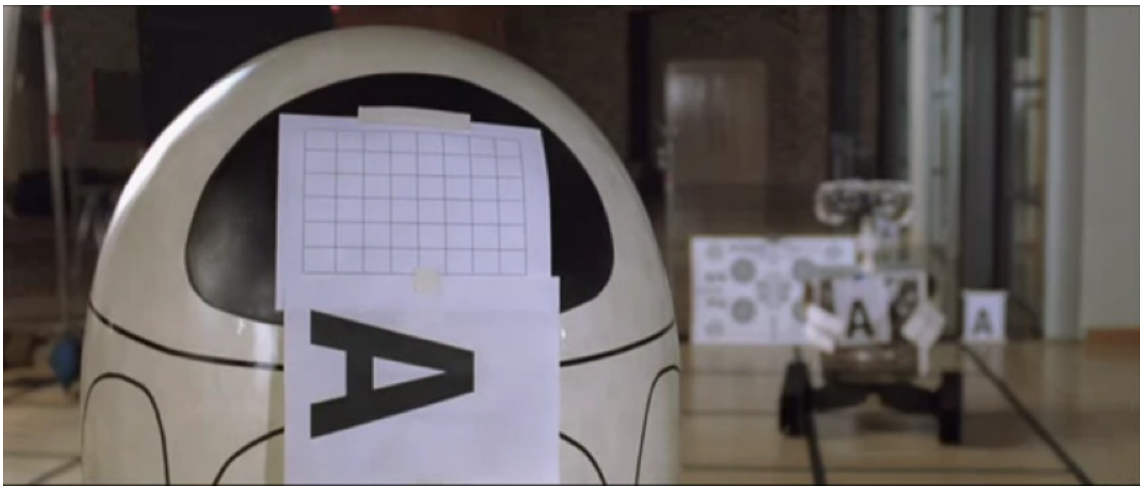

Figure 2.3: Cardboard stand-ins of characters Wall-E and Eve were used to perform lens tests for the movie *Wall-E*. [36]

2.2.2 Wall-E (2008)

Pixar Animation Studios broke new ground in the realm of camera control with their 2008 feature film *Wall-E*. In the film, Wall-E, a mostly-mute robot protagonist, goes about his daily duty of cleaning up an abandoned Earth, until he meets a new robot named Eve and gets thrust into an extra-terristrial adventure. For the first part of the film, director Andrew Stanton told director of photography Jeremy Lasky that he wanted the audience to believe that they were truly watching a little robot going about his business. The inspiration for this feel was rooted in live action cinematography, so Laskey and his team rebuilt Pixar's years-old camera package from the ground up to emulate the way a 35mm anamorphic camera would work. They rented equipment, performed lens tests with cardboard Wall-E and EVE stand-ins, studied lens flares, and consulted with live-action director of photography Roger Deakins, A.S.C, who had recently shot the live-action film *No Country for Old Men* [22]. They then poured all this knowledge into their new virtual camera system.

They made simple changes like moving the pivot point of the camera back from where the lens would be. They made more radical changes like scrapping their tried-and-true method for depth of field, replacing it with new-and-improved code. The end result of these changes was an "imperfect" system that generated more natural and realistic camera action. Using this new system, and the eyes of visual consultant Dennis Muren, A.S.C, Pixar made *Wall-E* feel as though it had been shot on a camera operator's shoulder [22].

Figure 2.4: Robert Zemeckis works with a virtual camera operator on the virtual set of *The Polar Express*. [23]

2.2.3 The Polar Express (2004)

In 2004, director Robert Zemeckis produced and directed a fully computed-generated (CG) film adaptation of the children's book *The Polar Express*. Though the entire film was CG, all the character performances were created using state-of-the-art motion capture (mo-cap) technology. Using arrays of special cameras and tracking markers, actors would have their movements and even their facial expressions recorded. These performances were then applied to the CG characters in the film. When it came to virtual cinematography, Zemeckis wanted to avoid the "too

perfect" feel that traditional keyframed virtual camera motion produced. To accomplish this, Ken Ralston and Jerome Chen, the co-senior visual effects supervisors on the film, developed a system they called "wheels." This system was essentially a computer device that simulated the feel of a real-life camera gearhead and could be operated by a real cinematographer using traditional techniques. In addition, they developed a series of virtual lenses that mimicked the performance of real 35mm optical lenses. To use the system, a previously recorded performance capture would be loaded into a virtual scene. The scene was triggered, and as the performance played out, the "wheels" operator could see the CG scene in his viewfinder, and was able to move around the scene and follow the characters just as though he were shooting live-action. Once the scene had finished, the camera motion could be saved and used for rendering the scene. This gave the final scenes the realistic cinematic feel that Zemeckis wanted [5].

Figure 2.5: James Cameron uses one of the virtual camera control rigs on the set of *Avatar*. [34]

2.2.4 Avatar (2009)

The 2009 film *Avatar* produced one of the most advanced virtual camera control systems to date. Like *The Polar Express*, *Avatar* used advanced motion capture technology to record the performances of actors for many of the characters in the film. Unlike the post-production method of generating virtual camera motion used in *The Polar Express*, director James Cameron wanted to be able to interact in real-time with his mo-cap actors as they were performing while also simultaneously viewing the CG environment they would be placed in. Glen Derry, the virtual-production supervisor, developed a two-part system to allow Cameron to do this. First, for scenes in which there were only CG characters present, Cameron would use a hand-held rig which had its position tracked in space to control the virtual camera. Looking at a monitor on the rig, Cameron could see the mo-cap actors as the CG characters they were playing in real-time. The second part of the system, dubbed SimulCam, was used when there were both live-action actors and CG characters in the same scene. In these situations, Cameron would use a live-action camera that was tracked by mo-cap sensors. A virtual camera in the CG environment would then be matched in position to this live-action camera, and the CG environment and characters would be composited into the live-action photography. The final composite image could be viewed in real-time through the SimulCam's viewfinder and on live monitors on set. This allowed the real-life actors to interact with the CG characters, and allowed Cameron to frame up the exact shots that he wanted without having to imagine how they would look later [26].

2.3 Motion-Sensing Systems

Like several of the production-specific setups mentioned in the previous section, our system for virtual camera manipulation includes physical components as part

of its design. A number of commercially-available devices exist that are capable of detecting motion. In this section, we investigate a few of these devices, focusing on how they operate, what kind of data they provide, and what their strengths and weaknesses are.

2.3.1 Accelerometers

An accelerometer is an electromechanical device that measures acceleration forces. The forces can be static, like the constant force of gravity, or they can be dynamic, caused by moving or vibrating the accelerometer. By measuring the amount of static acceleration due to gravity, one can determine the angle of the device's tilt with respect to the earth. By sensing the amount of dynamic acceleration, one can analyze the way the device is moving [12].

There are many different ways accelerometers can sense acceleration, such as using the piezoelectric effect, the piezoresistive effect, hot air bubbles, and even light [12]. Regardless of which method specific accelerometers utilize, they all generally function in a similar way.

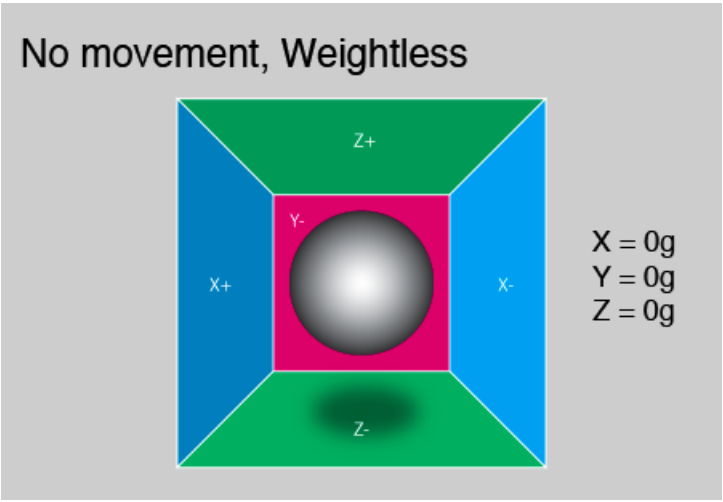

Figure 2.6: A hypothetical accelerometer being static in a weightless environment. [8]

When thinking about accelerometers, it is helpful to imagine a hypothetical hollow cube with a ball inside it. Each dimensional axis, x, y, and z, is assigned an opposing pair of faces in the cube. One face of each pair represents the positive direction of the assigned axis, and the other the negative. If the cube is taken to a hypothetical location with no gravity and no other fields that might affect the ball's position, the ball floats in the middle of the cube. An example of this setup and state can be seen in Figure 2.6.

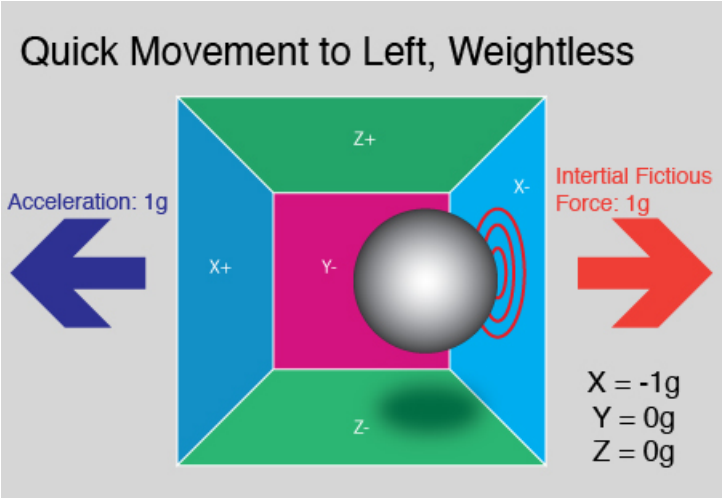

Figure 2.7: A hypothetical accelerometer that has been moved quickly to the left in a weightless environment. [8]

Now imagine that each face of the cube is pressure sensitive. If the cube pictured in Figure 2.6 is suddenly moved to the left at a rate of 1g (9.8m/s squared), the ball hits the wall X-, as seen in Figure 2.7. The pressure force that the ball applies to the wall is measured and the accelerometer outputs a acceleration value of -1g for the X axis.

It is important to note that the force an accelerometer actually detects is directed in the opposite direction from the acceleration vector. The force detected is not a "real" force, but rather an inertial force, or "fictitious" force. An inertial force is

an apparent force that occurs when motion is described within an accelerated frame of reference [37]. If the motion was described in the "frame of the universe", or one moving uniformly with the rest of the universe, these inertial forces would not appear. The centrifugal force felt when riding a "rotating drum" carnival rode is an example of an inertial force. In our hypothetical accelerometer, acceleration is measured indirectly through a force that is applied to one of its walls. (In a real accelerometer, the force-sensitive component is likely different.) This force can be caused by the acceleration, but that is not always the case.

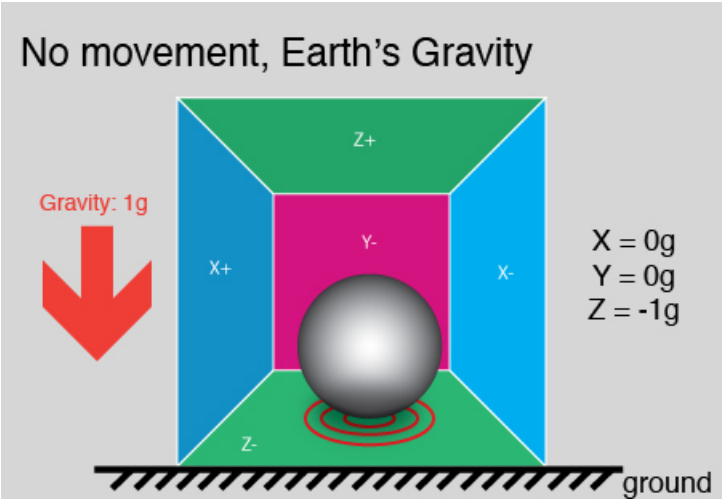

Figure 2.8: A hypothetical accelerometer that is sitting on the ground and experiencing Earth's gravity, but is otherwise static. [8]

Now imagine our hypothetical accelerometer is put on Earth. Due to Earth's gravity, the ball falls on the Z- wall and applies a force of 1g, as seen in Figure 2.8. In this case the cube isn't moving, but we still get a reading of -1g on the Z axis. The pressure applied to the cube's wall is not caused by acceleration of the cube itself, but by a different force. In theory the force doesn't have to come from gravity - for example, if the ball was metallic, placing a magnet next to the cube could move the ball so it hits a face of the cube. This would register a force that was caused by

neither gravity nor other accelerations. The important point to take away from this example is that in essence, an accelerometer measures force not acceleration. It just happens that acceleration can cause an inertial force that is captured by the force detection mechanism of the accelerometer.

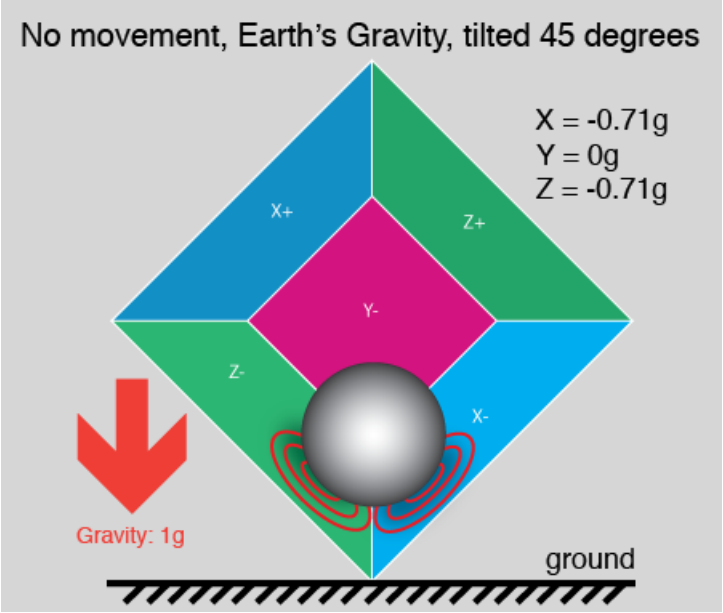

Figure 2.9: A hypothetical accelerometer that is sitting on the ground, is experiencing Earth's gravity, and is tilted 45 degrees clockwise around the Y axis. [8]

For our final example, we keep our hypothetical accelerometer on Earth, and tilt it 45 degrees to the right, around the Y axis. Due to Earth's gravity, the ball again falls towards the ground. But now, since the cube is tilted, the ball hits both the X- wall and the Z- wall, as seen in Figure 2.9. Each wall is detecting a portion of the force exerted by gravity. (Note: The values of -0.71g are actually approximations of -SQRT(0.5g). The manner in which these values were calculated will be discussed during the methodology section of this thesis.) If the device is not subject to any other forces besides gravity, we can use the data output by the accelerometer to calculate the inclination of the device relative to the ground plane.

2.3.2 Gyroscopes

A gyroscope is a device used for measuring or maintaining orientation, based on the principles of angular momentum [28]. Gyroscopes come in many different variations. One of the most widespread types of gyroscopes in use today is the Micro Electro-Mechanical System gyroscope, or MEMS gyroscope. A MEMS gyroscope is a specific implementation of a more general classification of gyroscopes know as vibrating structure gyroscopes or Coriolis Vibratory gyroscopes. The small, inexpensive MEMS gyroscope can be found in many modern cell phones, the Wii Motion Plus add-on accessory for the Nintendo Wii Remote, and many image stabilization systems utilized by digital cameras [24].

A MEMS gyroscope can measure how fast the rate of rotation is changing about a defined axis. To clarify with an example, let's say we have measured the rotation angle about a defined axis X at time t_0. We define that angle as Ax_0. Next, at a later time t_1, we measure the rotation angle again and find it to be Ax_1. The rate of change is calculated as follows:

$$Rate = (Ax_1 - Ax_0)/(t_1 - t_0)$$

If we express Ax_0 and Ax_1 in degrees, and t_0 and t_1 in seconds, then $Rate$ is expressed in degrees per second. This is the type of data that a MEMS gyroscope can provide.

It is important to further clarify what data the gyroscope provides by distinguishing between "change in rotation over time" and just simply a "change in orientation." A gyroscope does not, by itself, track how far it has rotated from a specified point, but rather how quickly its rotation is changing when it is in the midst of rotating. So if a gyroscope is rotating at a constant rate, the data it outputs do not change. Only when the speed of rotation changes, either by accelerating or decelerating, does

the data output of the gyroscope change. If we are able to measure the amount of time over which a rotation took place, we can multiply that amount of time by the data the gyroscope provides us to get the actual distance rotated. By itself, though, the gyroscope cannot directly tell us what our current orientation is.

Like the accelerometer, a MEMS gyroscope can use a variety of different internal setups to detect change in rotation over time, including tuning forks, vibrating wheels, and resonant solids [28]. Though there are several different implementations of MEMS gyroscopes, they all rely on the same principle: vibrating objects undergoing rotation.

A MEMs gyroscope has some form of oscillating component from where the acceleration, and hence direction change, can be detected. Per Newton's conservation of motion law, a vibrating object wants to continue vibrating in the same plane, and any vibrational deviation can be used to derive a change in direction [27]. In a gyroscope, these deviations are caused by the Coriolis effect.

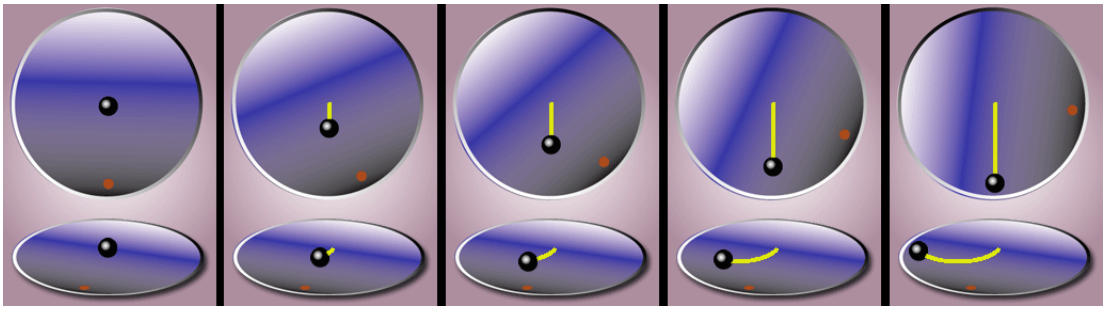

Figure 2.10: A visual example of the Coriolis effect at work. [13]

In physics, the Coriolis effect is the deflection of moving objects when they are viewed in a rotating reference frame [2]. A visual example of the Coriolis effect can be seen in Figure 2.10. In the inertial frame of reference (upper part of the picture), the black ball moves in a straight line. However, the observer (red dot) who is standing in the rotating/non-inertial frame of reference (lower part of the picture)

sees the object as following a curved path due to the Coriolis effect and centrifugal forces present in this frame of reference. If you have ever seen a Foucault pendulum in a museum gradually knock over pegs throughout the course of a day, you have witnessed the same thing that is happening in Figure 2.10. In this situation, the black ball is the pendulum, the blue circle is the earth, and you are the red dot. Due to the relative sizes of the earth and the pendulum, the Coriolis effect is not nearly as pronounced.

Figure 2.11: A Foucault pendulum. [42]

In simple terms, a MEMS gyroscope operates similar to a Foucault pendulum, except on a much smaller scale. When the gyroscope undergoes a rotation, the path of the vibrating element in the gyroscope undergoes a change, thanks to the Coriolis effect. This change is detected, and is reported via a change in data output. That data can then be interpreted to discover the change in rotation over time of the gyroscope.

2.3.3 Magnetometers

A magnetometer is a measuring instrument used to measure the strength and, in some cases, the direction of magnetic fields. At a basic level, it functions much like a standard, handheld compass. A compass consists primarily of a needle made of ferrous material suspended in such a way that it is free to swing around. The strongly magnetic needle can then rotate on its mount to align itself with the earth's magnetic field. Since these magnetic fields occur all across the earth, a compass can tell its owner which direction they are facing anywhere on the planet (with the exception of the magnetic poles).

Figure 2.12: A standard handheld compass. [16]

A magnetometer can sense the same magnetic fields as a standard compass, however it does not utilize a tiny, suspended needle to do so. Instead, it uses magnetoresistive sensors. How these sensors work is quite technical, but having a full understanding of their operation is not necessary to use them. It is sufficient to understand that the influence of magnetic fields on these sensors adjusts the electrical current flow through them. By applying a scale to this current, we can measure the amount

of magnetic force being experienced [9]. With two or more sensors on aligned axes, we can use the difference in the magnetic force detected by the sensors to infer our bearing to magnetic north.

Magnetometers do have a few limitations. Radio waves or magnetic fields other than the earth's can interfere with a magnetometer's operation. For instance, computers, speakers, mobile phones, and microwave ovens are just a few things that can cause interference if they are close enough, which affect a magnetometer's readings [9]. In addition, a magnetometer's reading of the earth's magnetic field varies depending on what part of the earth you are on. Magnetic north is determined by the earth's magnetic field, which changes over time and is not the same as true (or geographic) north. This phenomenon is called magnetic declination, which is defined as the difference between true north (the axis around which the earth rotates) and magnetic north (the direction the needle of a compass points)[32]. Luckily, this can be accounted for with minimal effort, provided the gyroscope is remaining in approximately the same location on earth during operation.

The most significant limitation of a magnetometer is that it only functions properly if it is on a level surface. If a magnetometer becomes tilted, it starts to become more and more inaccurate the further it strays from being level. With a three-axis magnetometer, it is possible to compensate for tilt. However this is only possible if the device's orientation with respect to the ground is knowable, which is data a magnetometer alone cannot provide.

3. METHODOLOGY

Rotational motion capture is the process of recording the movement of an object as it undergoes changes in orientation about an axis. The data recorded can then be used for other purposes. This thesis presents a system for physically-based, what-you-see-is-what-you-get virtual camera control by means of rotational motion capture. Unlike traditionally-used setups where the process of controlling the virtual camera is highly abstracted, the system presented in this thesis uses a control scheme where the input (physical gesturing) is directly correlated to the the output.

Two important corollary features of this system are the open, affordable nature and extensibility of both its physical parts and programming code. The system is intended to be easily modifiable by the user so it can be adapted to the specific needs of a particular project or production.

The approach here can be divided into two interacting components which allow the user to achieve their camera control goals. The first component is the data production method, which is the technical core of the approach and defines how the rotational motion data is created, detected, processed, and delivered as usable camera rotation information. The second component of this approach is the conceptual core of this thesis. The primary goal is defining the approach to artistic control of camera motion and the ability of the user to utilize and expand on the system in a number of ways.

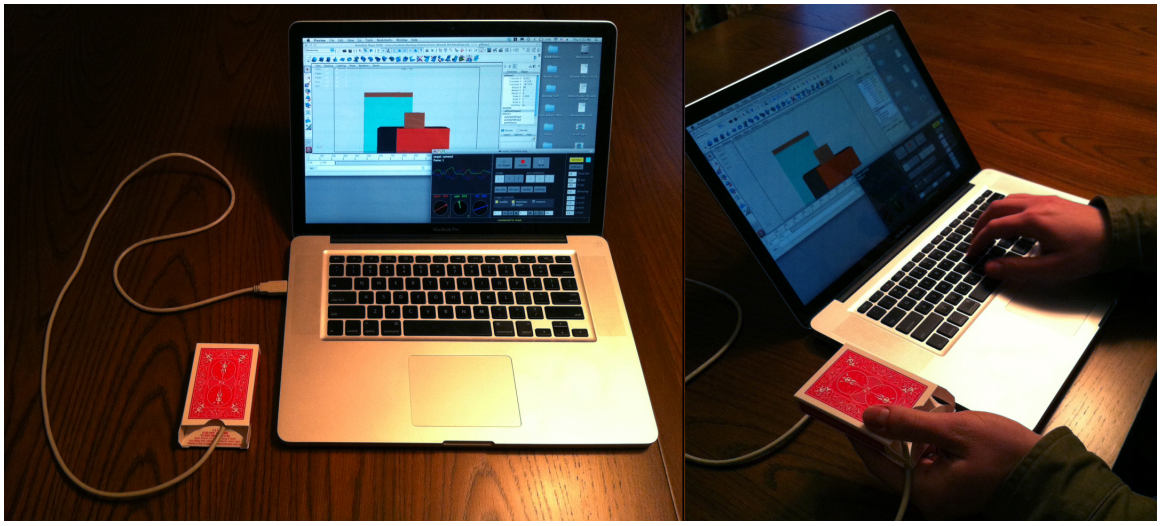

Figure 3.1: The proposed setup for the system. The physical device is represented by a red pack of cards. Autodesk Maya is open in the top left of the screen, and a mockup of the software application is open in the bottom right of the screen.

To use this system, the user needs to have the physical device and software application from this thesis as well as a few pre-existing items. The user must have a computer workstation with both Autodesk Maya and the software application from this thesis installed. The computer must have a mouse, keyboard, monitor, and a USB port. The user also needs a USB cord with a minimum length of three feet. The USB cord must be male USB B-type to male USB A-type. Finally, the user must have a workspace with enough room to comfortably accommodate their computer workstation and provide ample room for movement of their arms.

Once the user has gathered all the necessary items, they are ready to begin using the system. The user sits in front of the computer workstation, which is running both Autodesk Maya and our system's software application. The physical device is plugged into the computer via USB port, and should be set to the side next to the computer until needed. In Maya, the user opens a scene they would like to work

in. The scene should ideally be a low-resolution or reduced complexity scene such as a layout department would use, as heavy or complex scenes can cause Maya to become slow or unresponsive. The user selects or creates a camera they would like to capture motion for and then sets their main Maya viewport to look through that camera. When the user is ready to begin motion capture, they pick up the physical device with one hand and hold it in their desired starting orientation. They then connect the software application to Maya with a mouse click. Once the connection is successful, they move the physical device with one hand and view the results in real-time in the Maya viewport. While holding the device in one hand, the user uses their free hand to use mouse clicks or application hotkeys to adjust settings in the software application. A mock example of this setup can be seen in Figure 3.1.

It is important to note that the user experience setup does not seek to emulate the total user experience of controlling a real film or video camera. Film cameras are typically mounted on the operator's shoulder or a sophisticated rig, and the scene is viewed through a viewfinder or display that is attached to the camera itself. Having a shoulder-mounted device with a viewfinder could potentially add additional realism to the motion captured by our system. The additional weight and resistance coupled with manipulating the device when it is in a more "traditional" position for camera operation would offer a level of physicality that sitting in front of a computer screen can not.

3.2 Data Production Method

This approach introduces a system that utilizes multiple different sensors working in conjunction to detect rotational motion and output accurate data representing that motion. Using multiple sensors allows for redundancy and robustness, where each different sensor's weaknesses are accounted for by another sensor's strengths.

Based on this model, this thesis presents a data production method that consists of four stages:

- hardware selection
- sensor initialization
- data interpretation
- data output

3.2.1 Hardware Selection

The first stage of the data production method requires choosing sensors and other hardware with which it is possible to produce and interpret rotational motion data. Hardware selection is important, as the components chosen must allow for usable motion capture while at the same time remaining affordable and allowing for easy expansion of the device's functionality, should the user require it. In this method, we have decided to use a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer in conjunction with an Arduino board and microprocessor.

In-depth explanations for each of the three sensor types can be found in the Prior Work section.

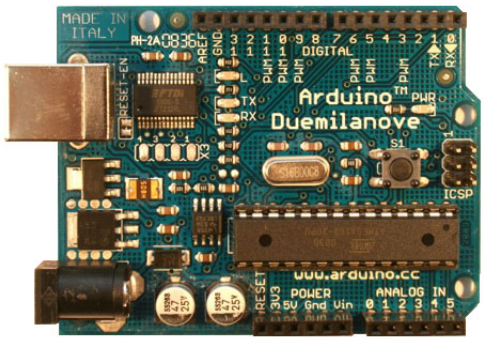

Figure 3.2: One version of Arduino, the "Duemilanove" board. [11]

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. Arduino is intended for artists, designers, hobbyists and anyone interested in creating interactive objects or environments [11]. In more technical terms, an Arduino consists of a simple open source hardware board designed around an 8-bit Atmel AVR microcontroller, or a 32-bit Atmel ARM [10]. The software consists of a standard programming language compiler and a boot loader that executes on the microcontroller. Multiple different variations of the Arduino hardware exist, each with a slightly different setup.

While certainly not the only option, Arduino was chosen for this method due to its ease of use, ample documentation, and versatility. It is very simple to get started using Arduino, even without prior experience in electronics. Arduino takes care of a large part of the very basic setup, allowing the user to jump in and begin working on their own project without having to reinvent the wheel, so to speak. For instance, most Arduino boards include a USB and power connection, allowing the user to easily power their device and attach it to their computer for code upload with little to no additional work. When it comes to writing code, Arduino uses a language very similar to Java and provides its own interactive development environment. Both the code and the interactive development environment (IDE) are cross-platform (they can be used on Windows, Mac OSX, and Linux). There are extensive tutorials and documentation on the official Arduino website, and there is an active Arduino development community that has been around for many years that can offer additional support and examples. Finally, both Arduino's software and hardware are open source. The hardware's original design files are available for free and licensed under the Creative Commons Attribution Share-Alike license, which allows for both commercial and personal derivative works as long as Arduino is credited [3].

For the purposes of this method, we can think of the Arduino board as the "brain"

of our device and the accelerometer, gyroscope, and magnetometer as being the "nervous system." The various sensors can only output streams of data; they cannot interpret the data or perform calculations by themselves. The sensors interface with Arduino, which, with the proper programming, interprets the data sent to it into a usable output.

3.2.2 Sensor Initialization

Once the Arduino board and the sensors are properly interfaced, it is necessary to initialize the sensors so that they function as we want them to. Each type of sensor has a set of attributes that affect how it operates. The specific attributes vary depending on the sensor. Each sensor also has a specific communication protocol, which dictates how data is sent to and accessed from the sensor. Most sensors fall into one of two categories: digital or analog. Digital sensors communicate using a serial protocol like I2C, SPI, or USART. Analog sensors output a voltage level within a predefined range, which must then be converted to a digital value using an Analog to Digital converter, or ADC [1].

To initialize a sensor, we have to read from and write to its registers. Registers are basically small amounts of storage available on a device or processor [15]. The sensors used in this method typically have several registers consisting of around eight bits, or one byte, each. Registers can be used to control attributes of the sensor. Usually some registers are used in conjunction to temporarily hold output data to be read from an outside source. Information about the registers for a specific electronic component can be found in the peripheral's data sheet, which is typically provided by the manufacturer.

Regardless of specific type or manufacturer, most of the sensors this method is concerned with typically have specifications for measurement range, sensitivity,

power or operation mode, and data rate.

SPECIFICATIONS

T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V, acceleration = 0 *g*, C_S = 10 µF tantalum, $C_{I/O}$ = 0.1 µF, output data rate (ODR) = 800 Hz, unless otherwise noted. All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.

Table 1.

Parameter	Test Conditions	Min	Typ[1]	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range	User selectable		±2, ±4, ±8, ±16		*g*
Nonlinearity	Percentage of full scale		±0.5		%
Inter-Axis Alignment Error			±0.1		Degrees
Cross-Axis Sensitivity[2]			±1		%
OUTPUT RESOLUTION	Each axis				
All *g* Ranges	10-bit resolution		10		Bits
±2 *g* Range	Full resolution		10		Bits
±4 *g* Range	Full resolution		11		Bits
±8 *g* Range	Full resolution		12		Bits
±16 *g* Range	Full resolution		13		Bits
SENSITIVITY	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	All *g*-ranges, full resolution	230	256	282	LSB/*g*
	±2 *g*, 10-bit resolution	230	256	282	LSB/*g*
	±4 *g*, 10-bit resolution	115	128	141	LSB/*g*
	±8 *g*, 10-bit resolution	57	64	71	LSB/*g*
	±16 *g*, 10-bit resolution	29	32	35	LSB/*g*

Figure 3.3: Part of a typical specification sheet for an accelerometer. [7]

Measurement range is the level or threshold of input supported by the device. A larger measurement range can handle and detect greater levels of input, but is also less precise than a smaller measurement range. For accelerometers, range specifies the amount of acceleration or force detection supported by the device, and is usually specified in +/-g. For gyros, it specifies the level of angular velocity, usually in units of +/- degrees per second or rotations per minute (RPM). For magnetometers, it is the strength of a magnetic field, and is usually measured in gauss [1]. Some devices allow for multiple measurement ranges, in which case the user must choose the one best suited for their needs.

Sensitivity is the ratio of change in input to change in output signal. A higher sensitivity means for a given change in input (acceleration, angular velocity, magnetic field, etc.), there is a larger change in the output signal, which is easier to measure and allows detection of more subtle movements. Sensitivity is specified at a particular operating voltage. It is usually measured in units of millivolts per measurement

range unit (g, degrees/sec, gauss) for analog output or Least Significant Bit (LSB) per measurement range unit for digital output [1]. Some sensors allow the user to choose from either a higher or lower sensitivity, depending on their needs.

Power or operation mode specifies whether the sensor is active or not. Different modes can include active, standby (off), and auto sleep (off after a period of inactivity). In addition, some sensors can be put into different operation modes. For instance, "normal" operation mode can sometimes mean a sensor reports its data once, then goes to sleep, whereas in continuous mode, the sensor is constantly reporting its data [1].

Data rate, also known as bandwidth, specifies the amount of times per second you can make a reliable data reading. It is measured in Hertz (Hz), and can vary in speed from hundreds of times per second to less than once every ten seconds.

For this method, we need to be familiar with each sensor's specifications. This allows us to choose the most appropriate settings for the purposes of our motion detection, and it helps to ensure that the sensors are working properly before any data collection or data processing is attempted.

3.2.3 Data Interpretation

When hardware selection and sensor initialization are complete, we can begin the process of converting the raw data output of our devices into usable rotational motion data in the form of pitch, roll, and yaw.

3.2.3.1 The Direction Cosine Matrix

Orientation kinematics typically deals with calculating the orientation of a body relative to a global coordinate system. For this method, we specifically want to calculate the orientation of our device relative to a starting orientation that we define ourselves. We call the frame of reference of our device the "body frame" and

the frame of reference of the starting orientation the "global frame." Both the body frame and the global frame have the same origin O, and have attached coordinate systems $Oxyz$ and $OXYZ$, respectively. We define i, j, and k to be unit vectors co-directional with the body frame's x, y, and z axes (such vectors are also known as "versors"). I, J, K are defined as the versors for the global frame.

Extending those definitions, we can say that in terms of body coordinates, vectors i, j, and k can be expressed as:

$$i^B = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, j^B = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, k^B = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Similarly, we can say the in terms of global coordinates, vectors I, J, and K can be expressed as:

$$I^G = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, J^G = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, K^G = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

With the definitions of our two frames in place, we can now define a Direction Cosine Matrix (DCM) for each frame. A DCM (also called a rotation matrix) can also be used to determine the global coordinates of a vector if we already know the coordinates in the body frame (and vice versa). In other words, it allows us to "convert" back and forth between body and global coordinates.

The DCM to convert from the body frame to the global frame can be defined as

follows:

$$DCM^G = \begin{pmatrix} I.i & I.j & I.k \\ J.i & J.j & J.k \\ K.i & K.j & K.k \end{pmatrix} = \begin{pmatrix} cos(I,i) & cos(I,j) & cos(I,k) \\ cos(J,i) & cos(J,j) & cos(J,k) \\ cos(K,i) & cos(K,j) & cos(K,k) \end{pmatrix}$$

where $I.i$ represents the scalar (dot) product of vectors I and i. Similarly, the DCM to convert from global frame to body frame is:

$$DCM^B = \begin{pmatrix} I.i & J.i & K.i \\ I.j & J.j & K.j \\ I.k & J.k & K.k \end{pmatrix} = \begin{pmatrix} cos(I,i) & cos(J,i) & cos(K,i) \\ cos(I,j) & cos(J,j) & cos(K,j) \\ cos(I,k) & cos(J,k) & cos(K,k) \end{pmatrix}$$

These two matrices are the transpose of each other, and are also orthogonal to each other.

To convert an arbitrary vector v^B that is represented in the body frame coordinate system to its global frame coordinate system representation v^G, we would perform the following calculation:

$$v^G = \begin{pmatrix} v_x^G \\ v_y^G \\ v_y^G \end{pmatrix} = \begin{pmatrix} cos(I,i) & cos(I,j) & cos(I,k) \\ cos(J,i) & cos(J,j) & cos(J,k) \\ cos(K,i) & cos(K,j) & cos(K,k) \end{pmatrix} \begin{pmatrix} v_x^B \\ v_y^B \\ v_z^B \end{pmatrix} = DCM^G v^B$$

And similarly, converting from v^G to v^B is simply:

$$v^B = \begin{pmatrix} v_x^B \\ v_y^B \\ v_z^B \end{pmatrix} = \begin{pmatrix} cos(I,i) & cos(J,i) & cos(K,i) \\ cos(I,j) & cos(J,j) & cos(K,j) \\ cos(I,k) & cos(J,k) & cos(K,k) \end{pmatrix} \begin{pmatrix} v_x^G \\ v_y^G \\ v_y^G \end{pmatrix} = DCM^B v^G$$

The specifics of how and why the DCM works is not discussed in this method, however a quick search online can provide numerous sources for further explanation. A good place to start online is: *http://mathworld.wolfram.com/DirectionCosine.html*.

In this method, we have two frames of reference with two different coordinates systems: the body (device) frame and the global (starting orientation) frame. The global frame remains static with respect to the earth. The body frame remains static with respect to the device itself. That means, for example, if we define a point on the device in the device's coordinate system as $r^B = (1, 0, 0)$, we can rotate the device any which way and that point remains $(1, 0, 0)$ in device coordinates. However, that same point expressed in global coordinates appears to change coordinates. Converting this fixed point r^B on the device into global coordinates using a DCM can tell us which direction the device is "pointing" in the global frame.

3.2.3.2 Defining and Updating the DCM

While we have established that a DCM matrix can convert any vector from the device's coordinate system (body frame) to the starting orientation coordinate system (global frame), this does not help us unless we are able to calculate the DCM matrix to begin with. By definition, at the start of our method the device orientation is the same as the starting orientation, so the DCM is initially defined as the identity matrix. After that initial instant, though, we are rotating the device, which means that the device's coordinate system is changing. This means that the DCM must be regularly updated to reflect the current relationship between the two coordinate systems. For this method, we use the sensors on our device to define and update the DCM.

As discused in the Prior Work section, a 3-axis accelerometer is capable of detecting tilt with respect to the force of gravity. The gravity vector consistently points

towards the center of the earth, perpendicular to the plane of the ground. However, because the accelerometer is also susceptible to outside forces besides gravity, and it has no way of distinguishing between different forces, its tilt readings are never be reliable enough to be usable by themselves.

The 3-axis gyroscope can tell us the angular velocity of the device. Angular velocity is defined as the rate of change of angular displacement. It is a vector quantity which specifies the rotational speed of an object and the axis about which the object is rotating. Unfortunately, gyros are subject to a phenomenon called "drift", where they gradually become more and more inaccurate over time unless they are calibrated by applying a calculated correction value. Because of drift, gyros alone cannot be used to accurately report rotational motion data.

The 3-axis magnetometer can tell us orientation with respect to magnetic north when held parallel to the ground plane. With this data, only rotation about the axis perpendicular to the ground plane can be calculated, and it is only accurate as long as the magnetometer remains parallel to the ground plane, as any tilt introduced throws off the readings. The magnetometer can also be thrown off by stronger local magnetic fields.

All three sensors (accelerometer, gyroscope, and magnetometer) provide different data regarding orientation. But each type of sensor also has weaknesses that prevent its data from being accurate or usable. The solution used in this method combines the readings from all three sensors in order to create an estimate or "best guess" of the device's orientation in the global coordinate system.

Gyroscopes have no sense of absolute orientation of the device. In other words, there is nothing with a known orientation that a gyroscope can compare itself to in order to determine its own orientation. The accelerometer and magnetometer, however, can compare themselves to the gravity vector and magnetic north, respectively,

to determine their orientation. The data from the accelerometer and magnetometer can be used to calculate a DCM, however the direct readings from each device are subject to a lot of noise in the form of external (non-gravitational) inertial forces or magnetic fields other than the earth's.

If we know the orientation of the device at time t_0, expressed as a DCM matrix DCM_0, we can calculate an updated DCM_1 at time t_1 using the gyroscope's angular velocity data. To prevent error accumulation due to drift, we "fine tune" the DCM using the accelerometer and magnetometer data, which are not subject to the drift that affects the gyros. In turn, the DCM defined by the gyro data can be compared to the accelerometer and magnetometer data to help filter out data noise. In this way, each sensor compensates for the other sensor's weakness, and we can compute a reliable and accurate DCM at each timestep.

3.2.4 Data Output

Once the device has calculated usable motion capture data, we need to be able to transfer that data from the device to a computer where it can be put to use.

Most Arduino boards offer a USB port that allows for serial data transfer. Serial transmission is the sequential transmission of signal elements of a group representing a character or other entity of data. Digital serial transmissions are bits sent over a single wire, frequency or optical path sequentially [14]. While certainly not the only option, we use it for this method since it is a very common data transfer method that is extremely versatile and has plenty of documentation available, making it easy to learn and to troubleshoot.

When transferring data, the most important considerations for this method are data format, speed, and accuracy.

Data format refers to how the information we are sending is "packaged" for

delivery. There are several options to consider. Data could be sent as multiple small packets of data, or as one large write that contains everything. Also, we must choose how much data to send. In addition to sending the core rotational motion information of pitch, roll, and yaw, we could send information about the raw data from all the sensors, or perhaps some of the data from the intermediate steps of our calculations.

In the context of this method, speed does not refer to baud rate of data transfer, but rather how often we are sampling new data and how often we are sending data updates. To determine speed, several things must be take into consideration. Traditionally, most films are shot at a rate of 24 frames per second. Since the data we are sending is designed to work as part of a movie pipeline, it is important that we are able to deliver updated camera motion data at a minimum of 24 times per second. Sampling data faster than 24 times per second can allow us to average multiple samples during a time step. As a simple example, if we sample our data 48 times per second, we can take use an average of every two samples for each of the 24 frames we actually want motion data for. Averaging data like this can help lessen the impact of noise and anomalous data readings. There are limits to how fast we can sample and send data, however. Each of the sensors has a limit on how fast it can update. Packaging data on the device end and then unpacking it on the receiving end also takes time. If the data rate is too fast and the data cannot be unpacked as fast as new data is being sent, the data may become corrupted.

We define accuracy as how correct data is when it arrives at its destination. Ideally our data transfer is completely accurate, with no data loss or corruption. As already touched on when talking about speed and data format, there are many ways for accuracy to be compromised. We must be careful when choosing our transfer speed and data format so we can maintain the integrity of our data. Because the

rotational data we are sending is being used in real-time, checksums were deemed unnecessary, as any errors in data would be recognizable in the form of very noticeable operational problems.

3.3 Artistic Control

The artistic control component of this method is the conceptual core of this thesis. Once the device is outputting usable data, the user needs to be able utilize the device in such a way as to achieve their camera control goals. There are several ways of facilitating this that we plan to implement.

3.3.1 Physical Device

The first and most obvious consideration is that the user be able to physically manipulate the sensor with relative ease. The device is not useful if the user is unable to effectively rotate it around to produce motion data. The device should be small enough and light enough to be operated by a single person with one or two hands. The construction of the device should be stable enough to allow for typical use without concern about damaging the device. For this prototype method, the device is attached to a computer via a USB cord. We must ensure that the USB cord is long enough to allow for mostly unimpeded motion during operation.

It is important to note that while we are using the device to capture motion that is used for a virtual camera, a true camera-like feel to the operation of the device itself is not the goal of this thesis. However, we strive to make the device small enough such that it can be mounted onto a real camera or a similar type rig setup, should that be the user's desire. It is important to note, however, that the utility of such a setup would be limited unless the Maya window could be displayed by the camera itself, and that is a non-trivial feature to implement.

3.3.2 Extensibility

Figure 3.4: An analog trigger control. [4]

The user must be allowed the ability to expand upon the basic rotational motion capture functionality of the device. To demonstrate this extensibility, in this method an analog trigger is added to the device. These triggers are the same ones found in modern game controllers. The most typical setup is to have a potentiometer that gets adjusted by a spring-loaded trigger [4]. Essentially the user can control the signal output by squeezing or relaxing the trigger to adjust its tension. In this method, the analog trigger allows for control of zoom or translation along the viewing axis, depending on the user's choice.

3.3.3 Software Application

To extend the versatility of the device, a software application is developed to allow for additional controls and relevant settings to be easily accessed by the user. The software serves as a gateway between the device and the 3D visualization software. This is a more flexible approach than having the device directly interface with the 3D visualization software. If the 3D software package being used changes, more code can be reused, and the programming of the device can remain consistent.

For this method, we use a language called Processing to develop the software application. Processing is based on Java and was developed by the same organization

that developed the Arduino. The IDE is very similar to the Arduino IDE, and it shares many of the same advantages, including being free, having a large amount of examples and documentation, and being multi-platform. While many different languages would have worked, Processing was chosen specifically for its close ties to Arduino, which should make it easier to learn and easier to use in conjunction with our chosen hardware.

The software application provides visual feedback regarding the operating state of the device. The user is able to tell if the device is active and whether or not it is currently connected to the 3D software package. There are numerical readouts of the pitch, roll, and yaw of the device that update in real time, as well as a visual representation of each axis that shows the tilt that is occurring in that axis on the device.

Controls are made available to the user in the software application that allows for both basic and advanced functionality. Basic functionality covers controls and functionality that are essential to normal operation, such as the ability to connect the software application to the 3D software package for data transfer. Advanced functionality either expands the user's options or increases their ease of use to allow for an optimized and efficient workflow. The user has the ability to turn on or off the streaming of data for pitch, roll, and yaw independently, or select a different operation method than simply 1-to-1 rotation interpretation. Playback controls (play, stop, rewind) and shortcut buttons, such as clearing all the recorded keyframes or returning to the beginning of the play range, save the user time and keep the number of times the user must operate within the 3D program itself to a minimum. The complete list of controls provided for this method can be found in the implementation section.

4. IMPLEMENTATION

The approach proposed in this thesis was implemented in a mostly modular fashion, such that any one part of the process is not too dependent on the specific implementation of another stage. This allows for more flexibility and easier adaptability.

The objective of the first part of the implementation is to assemble and program a device that accomplishes the data production method. The second part of the implementation focuses on facilitating the user's artistic and functional control by means of the software application and its accompanying workflow, controls, and options.

The implementation closely follows the concepts and process stages introduced in the Methodology section.

4.1 Data Production Method

The implementation of the data production method strives for low complexity and an easy learning curve. However, one of the core necessities of this approach is the ability to safely and effectively work with electronic components. Many hours were spent researching and learning basic concepts and terminology about electronics to establish a solid foundation of knowledge on which to begin our implementation.

4.1.1 Hardware Selection

For our sensor selection, we have chosen to use a device produced by Sparkfun Electronics called the SEN-09623 9DOF Razor IMU. The device incorporates an onboard ATmega328 microcontroller, which uses an 8MHz Arduino bootloader, and four sensors: a LY530AL single-axis gyro, LPR530AL dual-axis gyro, ADXL345

triple-axis accelerometer, and HMC5843 triple-axis magnetometer.

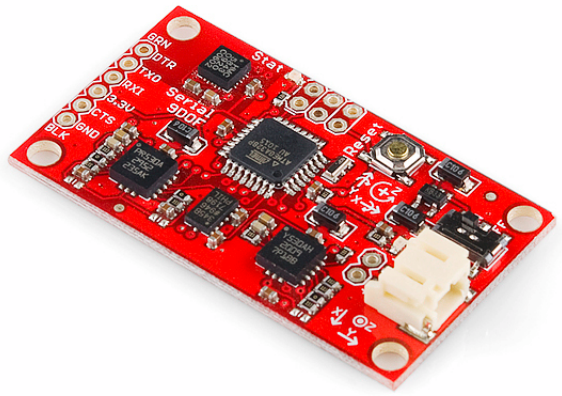

Figure 4.1: The Sparkfun Electronics SEN-09623 9DOF Razor IMU. [6]

This particular device was chosen for several reasons. The specifications of all the individual sensors incorporated into the board meet the operating requirements for our method. Having them all on a single board assures that the sensors' coordinate axes are aligned properly from the start and allows for a more compact and stable form factor. It assures that the sensors are wired properly, and keeps the amount of manual soldering we must do to a minimum. The addition of an on-board microprocessor programmed with the Arduino bootloader allows us to program the board to do calculations with data from all three types of sensors without needing to send the data to a different device. Functionally this allows us to treat the four sensors on the board as a single sensor unit, which strengthens the modularity of our approach.

We also make use of the Sparkfun Electronics COM-10314 Game Controller Trigger. While this sensor is not necessary for motion detection functionality, we are integrating it into our device to help demonstrate the ease with which the device can be expanded upon with custom controls. A picture of the trigger can be seen in Figure 3.4 in the methodology section.

We chose to utilize an Arduino Duemilanove board as the foundation of our device. The Duemilanove provides us with a USB port, which allows for serial transmission of data and for code upload to the microcontroller. The Duemilanove has sufficient inputs, both analog and digital, to interface with both the Razor IMU and the analog trigger, and includes a built-in analog-to-digital converter. A picture of a Duemilanove board can be seen in Figure 3.1 in the methodology section.

It should be noted that while we are using two microcontrollers for this implementation (one on the Razor IMU and one on the Duemilanove board), using only one would have been possible. However using two microcontrollers allows for a cleaner, less complex, and more modular setup where the individual motion-detecting sensors are compartmentalized as a single, mostly autonomous unit. With this setup we can make a clearer distinction between code written for motion-sensing purposes and code written for other functions. This additional ease of use and understanding is why the two-microcontroller implementation was chosen.

Because we have access to two microcontrollers, we must decide which calculations each one is responsible for. The Razor IMU is responsible for all calculations involved with converting raw sensor data into Euler angles (pitch, roll, and yaw). Once it has calculated Euler angles, its final responsibility is to package the Euler angles into a single sequence of characters, where each discrete data value is separated by a semicolon. This sequence of characters is then sent to the Arduino. The Arduino is responsible for performing calculations for any other extra components and controls added to the system. In our method, this means the Arduino is interpreting the data from the analog trigger. Once the calculations for the analog trigger are done, the Arduino appends the data from the analog trigger onto the end of the sequence of characters it received from the Razor IMU, once again separating each discrete data value with a semicolon. Finally, the Arduino sends this new sequence of characters,

which now contains both the Euler angles data and the analog trigger data, to the software application via a serial transfer over a USB cable.

The final setup for our prototype consists of the Razor IMU and Game Controller Trigger connected to the Duemilanove, and the Duemilanove connected to a laptop computer via a USB cord. Other minor pieces of equipment, such as precut wire, a breadboard, and a plastic enclosure for the device are also utilized in a supporting role. Some soldering is required, but our implementation keeps this task to a minimum.

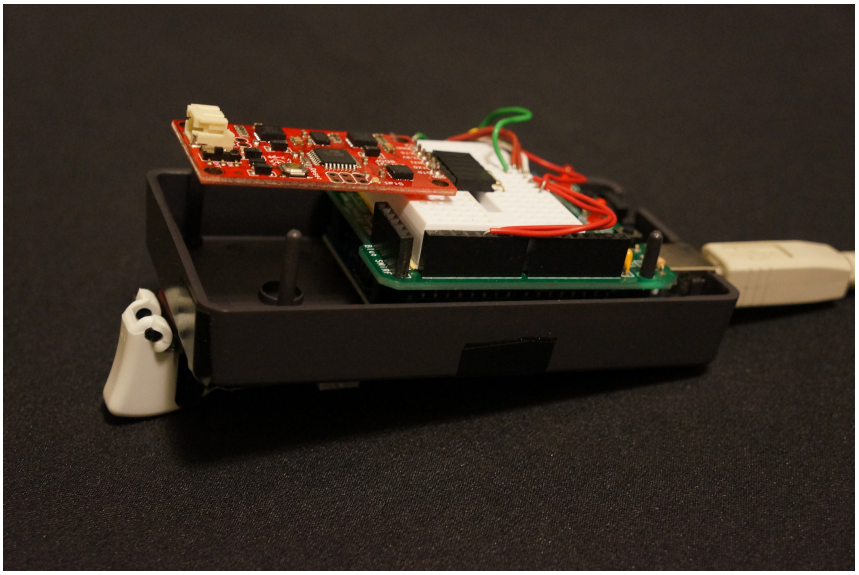

Figure 4.2: The physical device, pictured with the top of the plastic enclosure removed.

An illustration of the wiring setup used to connect all the components together can be seen in Figure 4.3 below.

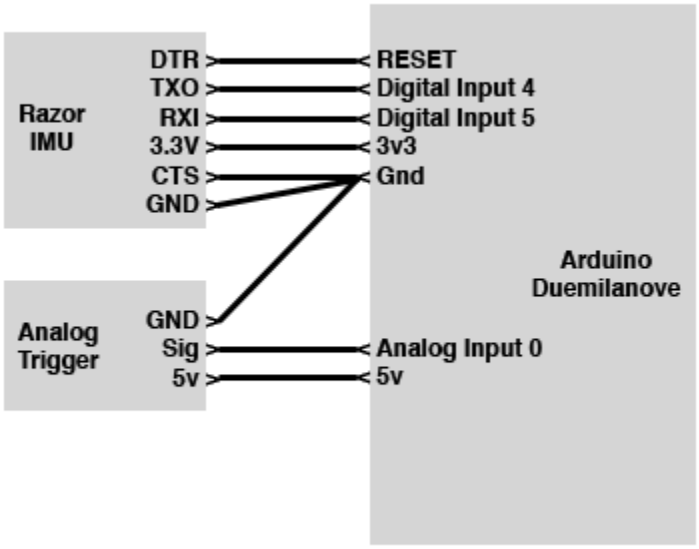

Figure 4.3: A diagram showing the connections between the different components.

Data sheets, schematics, and other specific information about the various peripherals used in this implementation can be found online at the Sparkfun Electronics website (for the IMU and analog trigger) and the Arduino website (for the Duemilanove board).

4.1.2 Sensor Initialization

For sensor initialization, the Arduino programming language and development environment were used. The development environment is available as a free download on the official Arduino website. It contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions, and a series of menus. It connects to Arduino hardware to upload programs and communicate with them. It was chosen for this implementation based on our hardware choices of the Arduino Duemilanove and the Razor IMU, which is programmed with an Arduino bootloader. For this implementation we specifically used version 0023 of the Arduino

development environment.

Before any sensor initialization was performed, the data sheets and specifications of each component were studied. This was done to assure understanding of what needed to be initialized for each component and how to perform the initialization.

For this implementation, if an attribute or setting of a specific component is not explicitly mentioned in the following subsections as being initialized, it is because the default value for that setting was determined to be appropriate for our needs.

4.1.2.1 ADXL345 triple-axis accelerometer

The ADXL345 triple-axis accelerometer is a digital accelerometer that uses the I2C communication protocol. The Arduino programming language includes a library called Wire that facilitates communication via I2C. It takes only a single line of code to include the Wire library in the project code and one additional line in the body of your code to initialize the library.

Once the Wire library is working, we initialize the ADXL345 itself. First we establish what the 7-bit address of the ADXL345 is so we know where to send the transmissions. (The address is 7-bit, as opposed to 8-bit, because the Wire library is designed to use 7-bit addresses.) According to the datasheet for the Razor IMU, the ADXL345's 7-bit address is 0x53. This number is represented in hexadecimal notation, as is the standard for register addresses.

We set the fourth bit of the ADXL345's power register (0x2D) to 1, which puts the accelerometer into measurement mode (a.k.a. "active" mode, the opposite of standby). In the data format register (0x31) we set the fourth bit to 1, which puts the accelerometer in full resolution mode, which makes the output resolution increase with the measurement range. In the rate register (0x2C), we set the output rate to 50Hz (0x09), which is considered normal operation.

Finally, we compute an offset, or "zero" value, for the accelerometer. This value is computed when the device is first activated, before the main loop begins to run. We take the average of 32 samples of the data for each axis from the accelerometer. This value is used as the "zero" value for the accelerometer; the orientation that reports this value is considered our zero-degree heading. When we get a raw reading from the accelerometer, we subtract this value as an offset to get our modified heading. This offset value can be reset during operation by the user if they desire to set a new zero heading.

The z axis of the accelerometer has one additional step involved when computing its offset. In a "neutral" orientation where the x axis and y axis are both reporting 0g, the z axis should be reporting -1g. If we simply computed an offset and subtracted it from the z axis's raw value, all three axes would report 0g at the neutral position, which is not what we want. To fix this, we subtract an additional 1g (approximately 248 raw data units) from the accelerometer's z axis when calculating its offset.

4.1.2.2 HMC5843 triple-axis magnetometer

The HMC5843 triple-axis magnetometer is, like the ADXL345, a digital component that uses the I2C communication protocol, so it also utilizes the Wire library for communication. The 7-bit address of the magnetometer is 0x1E. The only initialization required for our purposes is to put the magnetometer in continuous operation mode at a default rate of 10Hz, which we accomplish by writing zero (0x00) to the mode register (0x02). Otherwise the default settings for the magnetometer are sufficient.

4.1.2.3 LY530AL single-axis gyro and LPR530AL dual-axis gyro

Though we use two gyros in this implementation, their orientation with respect to each other allows us to think of them functionally as a single triple-axis gyro. Unlike

the accelerometer and magnetometer, the gyros are analog components, which means that an analog-to-digital converter (ADC) needs to be used to interpret the analog output.

The ADC must be initialized before it can be used. In order to accomplish this, it was necessary to do research to learn the setup and terminology of an ADC. After sufficient research, we were capable of initializing the ADC. To start, we set the ADC Enable bit (ADEN) in the ADC Control and Status Register A (ADCSRA) to 1, which enables the ADC itself. Also in the ADCSRA, the ADC Interrupt Enable bit (ADIE) was set to 1 to enable the ADC interrupt.

The ADC interrupt vector is essentially a block of code that is executed every time a conversion (from analog to digital) has been completed. In the context of our project, it is called every time the gyros' output has been converted into digital data. The ADC looks for a function named ISR to handle what happens during an interrupt. We can define this function to control its behavior.

In simple terms, our ISR function reads data from one of three "gyro" ADC channels (channels 0, 1, and 2 have the z, x, and y outputs, respectively, from the two gyros). It saves this data into a buffer to be used for calculations later, then increments some counters so that it reads and saves data from the next "gyro" channel. Finally, it signals the ADC that the next conversion can be started, and the function ends.

Once the ADC is taken care of, the gyro initialization is nearly complete. Like the accelerometer, our final task is to compute an offset for each axis of each gyro by taking the average of 32 samples of data before the main loop begins.

4.1.2.4 Razor IMU

There are a few attributes that must be set for the Razor IMU's microcontroller to operate properly. The "Analog_Reference" attribute configures the reference voltage used for analog input (i.e. the value used as the top of the input range). For our implementation we set this to DEFAULT, which in this context is equivalent to 3.3 volts. Next we must open a serial port for communication. The Arduino IDE provides a library called Serial to facilitate serial communication between an Arduino and a computer or other device. For the Razor IMU, we open a serial port and set our baudrate (a measurement of transmission speed) to 38400.

We must also initialize our Direction Cosine Matrix. Because we define our two coordinate systems as the starting orientation of the device and the current orientation of the device, at the very beginning of the device's operation before any movement has occurred, the two coordinate systems are perfectly aligned. Because there is no difference between the two coordinate systems at initialization time, we initialize the DCM to be the identity matrix:

$$DCM = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

4.1.2.5 Arduino Duemilanove

Like the Razor IMU, we must initialize serial communication on the Duemilanove. Unlike the Razor IMU, we must do it for two different serial data streams. First, we must open a port to the Razor IMU. We set this connection's baudrate to 38400 to match the Razor IMU. Next, we initialize a connection that goes from the Duemilanove to the computer via USB cord. The baudrate for this connection is set to

115200.

When we were initially setting the baudrate for the data stream between the Razor IMU and the Duemilanove, we tried setting it to 115200, which was the fastest option available. Later in our testing we noticed we were getting anomolies in the data being sent from the Razor IMU. At first we thought there was an error in the code, but after many tests we determined that this was not the case. After a large amount of internet searching and several visits to online forums, we discovered that sometimes if the baudrate is set too high, the receiving device cannot interpret the data quickly enough, resulting in corruption. This turned out to be true in our case, and reducing the baudrate between the Razor IMU and the Duemilanove to 38400 eliminated the data corruption we were getting.

Finally, we must make sure to set the specific type of Arduino board we are using in the IDE. If we do not do this, we get unexpected results when we attempt to upload our code to our Arduino from the IDE. We set the board we are using by going to the "Tools" menu, then to "Board", and then selecting "Arduino Duemilanove or Nano w/ ATmega328."

4.1.3 Data Interpretation

For data interpretation, version 0023 of the Arduino programming language and development environment is used. The majority of the data interpretation is performed on the Razor IMU. The microcontroller on the Razor IMU has a loop function that is run and rerun continuously. In simple terms, our loop function consists of reading raw data from each of the three sensors, transforming that raw data into a form we want, outputting the usable data, and then repeating the loop.

4.1.3.1 Compute the Timestep

The first step at the beginning of a new loop run is to compute the timestep. The timestep is how much time has passed since the last iteration of the loop was run. We accomplish this by reading the system time at the beginning of a loop run, saving it, and then reading the time again at the beginning of the next loop run. The difference between these two values gives us our timestep. The timestep is usually on the order of milliseconds.

4.1.3.2 Read Raw Data

We read output data from the gyros first. It is important to note that even though the gyros are technically being read before the other sensors, the entire loop executes in a matter of milliseconds. So while the sensors are not, strictly speaking, being read at the same time, the time disparity between the readings is small enough to be considered inconsequential for our purposes.

As discussed in the ADC initialization section of the implementation, every time the gyros output a new raw data value, an ADC interrupt occurs, and the data from the gyros is read, converted, and stored in a buffer. This occurs many times within a given timestep. When it is time to read gyro data in our loop, we access the data in this buffer. In order to get more stable and noise-free data, we calculate the average of all the data values reported during the timestep and use that as our final raw data value for the entire timestep. The gyro raw data buffer is then cleared to prepare it to receive new data values during the next timestep.

Next we read raw data from the accelerometer. Using the Wire library, we read 2 bytes of data for each of the accelerometer's axes (6 bytes total). Each set of two bytes represents a single data value stored in two's complement form. We must convert this data value to a typical decimal number by performing a logical left

shift by 4 on the "high" byte and then performing a bitwise OR between the two bytes. The final step is to subtract each axis's offset, which was computed during the initialization phase. Once that is complete, we save the data from each axis.

Because the magnetometer also uses the I2C protocol for communication, reading its raw data follows the same pattern as the accelerometer. We read 6 bytes of data, convert it to 3 decimal values, and subtract an offset. Unique to the magnetometer, we must also multiply all readings by -1 because the magnetometer's axes all oriented in the opposite direction from the alignment we need.

4.1.3.3 Tilt Compensation for Magnetometer

We must compensate for any tilt the magnetometer is experiencing before its data can be considered accurate. We use the pitch and roll values that we calculated from our previous timestep to tilt-compensate the magnetometer's data with the following equations [17]:

$$CorrectedMagX = RawMagX * cos(pitch) + RawMagY * sin(roll) * sin(pitch) + RawMagZ * cos(roll) * sin(pitch)$$

$$CorrectedMagY = RawMagY * cos(roll) - RawMagZ * sin(roll)$$

Now we have the x and y components of our magnetic vector in the x-y plane. We calculate our final, tilt-compensated heading with the equation:

$$MagHeading = atan(-CorrectedMagY, CorrectedMagX)$$

4.1.3.4 Update DCM

Now that we have all the raw data from our sensors, we begin to perform calculations to transform that data into a form we want, namely Euler angles, or pitch, roll, and yaw.

First, we transform the gyros' raw data into radians/second data, the standard form for angular velocity, using the formula below:

$$Rate = ((GyroRawData - Offset) * (Vref/ADCsteps))/Sensitivity$$

The $(GyroRawData - Offset)$ term has already been calculated previously when we read the gyro's raw data and subtracted our calculated offset. $Vref$ is reference voltage, which for our device is 3.3V or 3300mV. $ADCsteps$ is how many discrete values we can translate the voltage into. This is dependent on how many bits the ADC has to work with. In our case it has 10 bits, so $ADCsteps$ is 1023. $Sensitivity$ for gyros is measured in mV/degree/second. The datasheet lists the sensitivity as 3.33mV/degree/second, however our own tests show this value is much closer to 3mV/degree/second in practice, so we use that value. After performing the calculations, we are left with $Rate$, measured in degrees/second. As a final step we convert $Rate$ into radians/second by multiplying it by $(\pi/180)$. If we compute $Rate$ for each of the three axes, we can combine the values into a vector $\delta\theta$ that represents the angular displacement.

Next, we use $\delta\theta$ to update our DCM. We define the DCM from the previous

timestep as DCM_0, and we define the rows of DCM_0, from top to bottom, as vectors I_0, J_0, and K_0. We know these vectors represent versors of the global frame expressed in body frame coordinates at time $t0$. We want to calculate DCM_1 (with row vectors I_1, J_1, and K_1), which is the DCM at time $t1$ after a timestep of duration δt, or $t1-t0$.

We want to know the position of our vectors after a small period of time δt has passed. The equation for position is:

$$Position_1 = Position_0 + (Velocity * (time_1 - time_0))$$

Adapting the equation for our purposes and using an arbitrary vector V we get the equation

$$V_1 = V_0 + (v * (time_1 - time_0)$$

where v represents the linear velocity vector. The linear velocity vector can be calculated from angular velocity using the equation

$$v = \omega \times R$$

where ω is the angular velocity vector and R is the vector that is rotating [20]. Our gyro provides us with the data for ω, and R is simply our initial known vector. Substituting this data into the original equation for calculating V_1, and replacing $(time_1 - time_0)$ with δt, we get

$$V_1 = V_0 + (\omega \times V_0) * \delta t$$

We substitute our DCM vectors into this equation to calculate I_1, J_1, and K_1, which we then use together to form DCM_1. This is the core algorithm we use to

calculate the DCM at time $t1$ from the previous DCM at time $t0$. It is run every timestep and keeps the DCM updated. However a few additional steps are required before the updated DCM is completely ready to use.

4.1.3.5 Renormalize DCM

We know the vectors I_0, J_0, K_0 form a valid DCM matrix. In other words, they are orthogonal to each other and are unity vectors. We cannot say the same thing about the updated vectors I_1, J_1, and K_1. The formulas used for calculating them do not guarantee preservation of vector orthogonality or length. Because δt is very small, the vectors are not off by much, and it is possible to correct them for each timestep with a series of equations.

We first focus on correcting orthogonality of the three vectors. To understand the final equations used, it is helpful to explain starting with a simple case. We define two unity vectors a and b that are very close to being orthogonal. We want to find a "corrected" vector b' that is orthogonal to a and is in the same plane formed by the vectors a and b. To do this, we first calculate vector c with the equation

$$c = a \times b$$

By definition, a vector that is the result of a cross product is orthogonal to both original vectors. Therefore c is orthogonal to both a and b and is perpendicular to the plane they form. Next we calculate the vector b' with the equation

$$b' = c \times a$$

Once again, by definition of the cross product, b' is orthogonal to a and c. This means b' is in the plane perpendicular to c , which is of course the plane formed by

a and b. So b' is the "corrected" vector we want - it is orthogonal to a and exists in the plane formed by a and b.

If we substitute into our equation for b' using the triple product expansion [41] and knowing that a is a unit vector and therefore $a.a = |a| = 1$, we get

$$b' = c \times a = (a \times b) \times a = -a(a.b) + b(a.a) = b - a(a.b)$$

We can then consider the term $-a(a.b)$ to be our "error correction" value to make b orthogonal to a. In this scenario, we only corrected vector b and left a alone. In a similar scenario where we correct vector a and leave b alone, we get the equation

$$a' = a - b(a.b)$$

where the error correction value for a is $-b(a.b)$.

We've corrected for both vectors individually, but we need a solution that corrects them together. Since neither vector is "more correct", we can intuitively split the correction between both vectors and only apply half of each error correction value, as shown below:

$$a' = a - b(a.b)/2$$

$$b' = b - a(a.b)/2$$

It is important to note that performing these calculations does not prove mathematically that a' and b' are orthogonal. Intuitively, however, we know it gets the angle in between the two vectors closer to $\pi/2$ radians or 90 degrees. Because we are working with very small intervals, if we perform this correction every time step, it assures that the two vectors never stray too far from being orthogonal.

Using the idea just discussed, we correct our newly updated DCM (with row vectors I_1, J_1, and K_1). We begin by correcting orthogonality between I_1 and J_1 vector of the DCM with the following equations:

$$I_1' = I_1 - J_1(I_1.J_1)/2$$

$$J_1' = J_1 - I_1(I_1.J_1)/2$$

We then normalize I_1' and J_1' by dividing each vector by its magnitude:

$$I_{corrected} = I_1'/\|I_1'\|$$

$$J_{corrected} = J_1'/\|J_1'\|$$

Finally, to calculate $K_{corrected}$, we simply take the cross product of our other two corrected vectors:

$$K_{corrected} = I_{corrected} \times J_{corrected}$$

We use these corrected vectors to recompose our DCM.

4.1.3.6 Drift Correction

Every timestep we must correct for the gyro's drift. We do this by comparing the vectors in our DCM (which are defined by data from the gyros) to reference vectors defined by the data from the accelerometer and magnetometer. Our final "estimated" vector positions are weighted sums of the the vector positions given by all the sensors.

First we use our accelerometer data to correct drift of the DCM. This is possible because the accelerometer provides a reference vector (gravity) to which we can compare the orientation of our DCM's K vector. To start, we take the raw accelerometer

data we've previously collected, put it into vector form, and then find the magnitude using the standard formula for calculating magnitude of a vector. Once we have our raw magnitude value, we divide it by our known analog value for 1g, in this case 248. This gives us the accelerometer vector's magnitude in terms of gs.

The purpose of finding the magnitude of the accelerometer vector is to help estimate how reliable the data coming from the accelerometer is. When the only force acting on the accelerometer is gravity, the magnitude of the accelerometer vector should be very close to 1g. If additional outside forces are acting on the accelerometer, the magnitude with be more or less than 1g. The further from 1g the magnitude is, the more significant the outside forces are that are acting on the accelerometer, and the less reliable the accelerometer's data is.

To account for this, we calculate a reliability value for the accelerometer using the following formula:

$$reliability = constrain(1 - 2 * abs(1 - magnitude), 0, 1)$$

If *magnitude* is exactly 1g, then *reliability* is 1. If *magnitude* is greater than 1.5g or less than 0.5g, then *reliability* is 0. Any value in between returns a decimal number between 0 and 1.

Once we have our reliability value, we compare the accelerometer vector to the K vector of our DCM by taking the cross product. If the accelerometer vector and the DCM K vector are parallel (which indicates that both the DCM and accelerometer believe the device to be tilted in the same orientation), the cross product results in the zero vector. If they are not parallel, the cross product gives us our "error" vector. We multiple the error vector by the reliability value to get the $RollPitchCorrection$ vector.

Next we use our magnetometer data to further correct the DCM. This is possible because the magnetometer provides a reference vector (magnetic north) to which we can compare to the orientation of our DCM's I vector. The error correction is equal to the z-component of the cross product of the x (left) column of the DCM matrix and the $MagHeading$ vector we previously calculated when we tilt-compensated the magnetometer's data in section 4.1.3.3. We take $MagHeading$ and calculate just the x and y components of the vector:

$$m_x = cos(MagHeading)$$

$$m_y = sin(MagHeading)$$

Then we compute the equivalent of just the z-component of the cross product mentioned above to get our yaw correction value:

$$correction = (I_x * m_y) - (J_x * m_x)$$

Finally, we must multiply *correction* by the K vector of the DCM matrix to get our final correction for yaw:

$$YawCorrection = correction * K$$

$YawCorrection$ is added to $RollPitchCorrection$ to get $TotalCorrection$, which is the vector we use to correct for drift. To actually perform the correction, we simply add $TotalCorrection$ to the the angular velocity vector we computed from our gyro data before our next timestep. (It should be noted that that the true amount of error correction applied is determined using a Proportional-Integral (PI) correction

system. In a nutshell, a PI correction system helps control both immediate, large errors as well as smaller errors that build up over longer time periods.)

4.1.3.7 Convert to Euler Angles

Our final step in data interpretation is to convert our DCM data into Euler angles. We have previously established that to convert a vector from body coordinates v^B to global coordinates v^G we use the following equation:

$$v^G = DCM^G v^B$$

We can accomplish the same transformation by performing a series of three consecutive basic rotations on the vector, one for each axis. We define the yaw angle as ψ, the pitch angle as θ, and the roll angle as ϕ, and get the following equation:

$$v^G = \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix} v^B$$

It is important to note that the order in which the rotations are applied is not arbitrary. They must be performed in this order to get proper results. When we combine the 3 rotations into one matrix, we get:

$$\begin{pmatrix} cos(\theta)cos(\psi) & sin(\phi)sin(\theta)cos(\psi) - cos(\phi)sin(\psi) & cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) \\ cos(\theta)sin(\psi) & sin(\phi)sin(\theta)sin(\psi) + cos(\phi)cos(\psi) & cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi) \\ -sin(\theta) & sin(\phi)cos(\theta) & cos(\phi)cos(\theta) \end{pmatrix}$$

This matrix is, essentially, our DCM matrix expressed in terms of Euler angles. Therefore, in order to find the Euler angles themselves, we use the following equa-

tions:

$$Pitch\theta = -asin(DCM_{[2,0]})$$

$$Roll\phi = atan2(DCM_{[2,1]}, DCM_{[2,2]})$$

$$Yaw\psi = atan2(DCM_{[1,0]}, DCM_{[0,0]})$$

4.1.3.8 Analog Trigger Setup

Setting up the analog trigger is very straightforward. The component is essentially a 10k potentiometer with a range of about 2k to 9k ohm that gets adjusted by a spring-loaded trigger. We connect the ground and power wires from the trigger to the ground and power outputs of the Arduino Duemilanove. We connect the trigger's remaining wire to one of the Duemilanove's analog inputs. Reading data from the analog trigger requires only a single line of code implemented on the Duemilamove. The value we use is simply the raw data output from the analog trigger. It is processed later in the software application.

4.1.4 Data Output

Now that we have our data, we must send it out from the device so our software application can make use of it. We convert our pitch, roll, and yaw data from radians to degrees and then write the data to a stream using the Serial library of the Arudino IDE. The discrete data values are seperated by semicolons to allow for easy parsing of the data at the receiving end. This stream of data is sent from the Razor IMU to the Arudino Duemilanove. A baudrate of 38400 was chosen for this stream because during tests higher speeds resulted in the Duemilanove struggling to process the incoming stream fast enough, which led to data corruption.

Once the Duemilanove has successfully read the data stream from the Razor IMU, it adds the data from the analog trigger onto the end of the data stream. The new

data stream is once again output using the Serial library, this time at a baudrate of 115200.

4.2 Artistic Control

The implementation of the artistic control method strives for both general usability and plentiful options. Ideally the user should be able to acheive their camera control goals as quickly and efficiently as possible, without having to perform a large amount of manual setup.

The core of the artistic control implementation is the software application. We chose to use a language called Processing to develop our software application. Processing is based on Java and is developed by the same group that developed the Arduino. The IDE is very similar to the Arduino IDE, and it shares many of the same advantages, including being free, having a large amount of examples and documentation, and being multi-platform. While many different languages would have worked, Processing was chosen specifically for its close ties to Arduino. For this implementation we use Processing version 1.2.1.

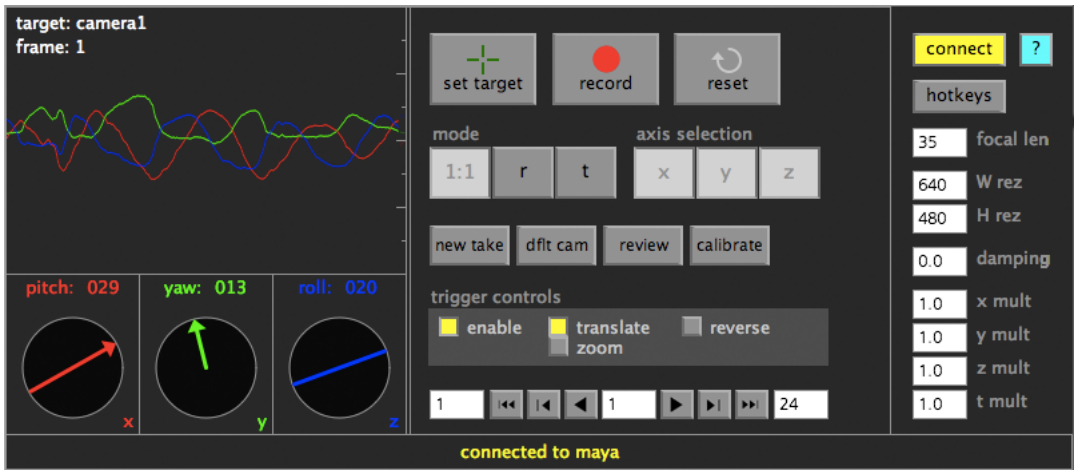

Figure 4.4: The software application.

The software application as a whole can be seen above in Figure 4.2. We explain

the functionality of each part in turn, and then provide a typical workflow for using the application.

4.2.1 Controls

Along the very bottom of the screen is the message center. When the user hovers over a button, a tooltip explaining the button's function is displayed here. The message center also lets the user know whether or not the are connected to a 3D software package, and shows when the recording button is active.

The connect button (upper right, yellow) allows the software application to connect to a 3D software package. For this implementation, it has been coded to connect to Autodesk Maya. The software application can function without being connected to Maya, however its functionality is limited.

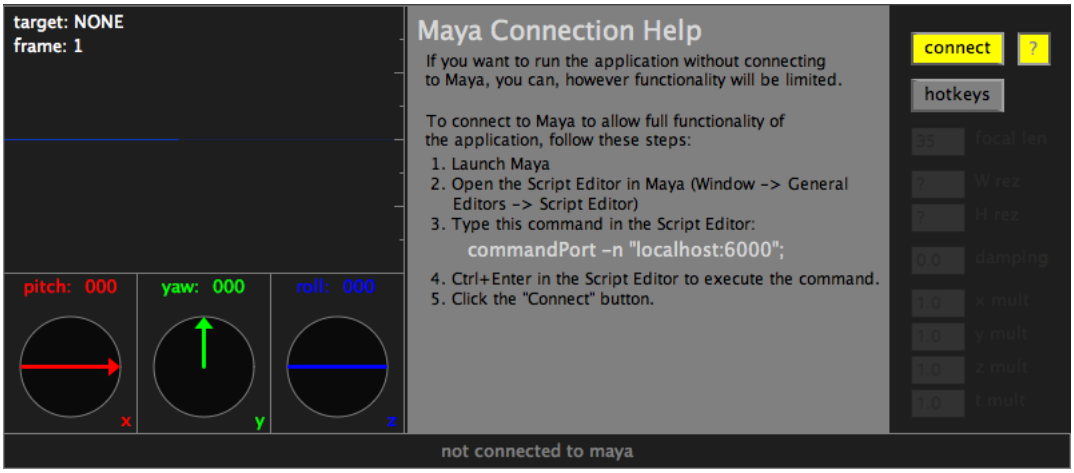

Figure 4.5: The software application showing the connection help screen.

Before a connection to Maya can be made, a command port must be opened in Maya. Clicking the question mark button (upper right, teal) brings up a set of brief instructions on how to set up a command port in Maya. This must be done before the connect button is pressed. Once a command port is established, it remains active as long as the Maya session is open.

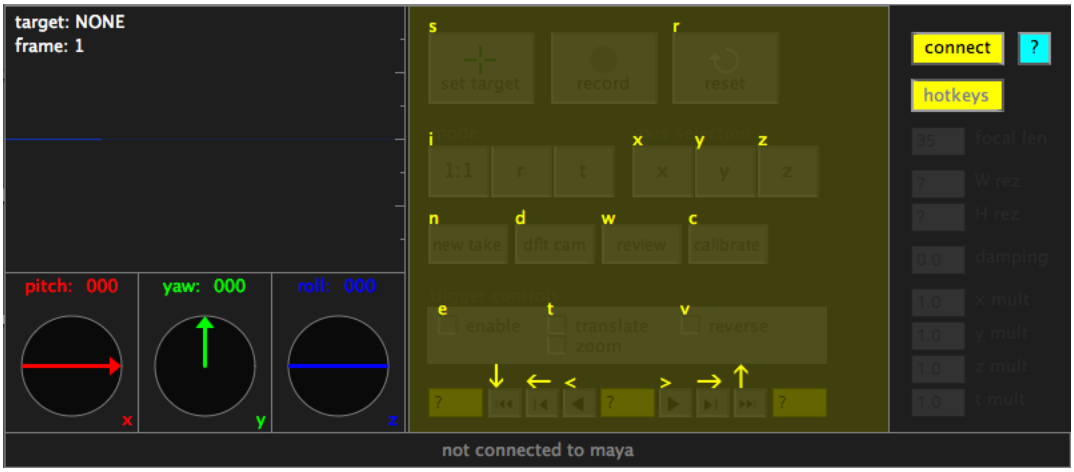

Figure 4.6: The software application showing the hotkeys interface.

The hotkeys button (upper right) brings up a screen that allows the user to set hotkeys for the majority of the available buttons. This allows for activation of buttons with a keypress as opposed to a mouse click, and allows for a more custom setup by the user. The current hotkeys are displayed above the buttons in yellow text. To set a new hotkey, the user first clicks on the button they want to set a hotkey for. The button's current hotkey changes to a question mark (?). The user then presses the key they want to be the button's hotkey, and it is assigned to that button. Hotkeys can be any letter, number, symbol, or any of the directional keys.

Down the right hand side of the interface are several textboxes. Users can change the values of the textboxes to affect the setting listed next to the textbox. Focal length controls the focal length of the currently selected camera object in Maya. W rez and H rez control the width and height dimensions of the output frame.

Damping is a value the user can set to help suppress noisy data. If the change in data from the previous state is less than the damping value, the application considers the data to not have changed. The result is less noisy output at the expense of precision. If a value of 0 is set for damping, all changes in value, no matter how

small, are considered valid. Values between 0.0 and 1.0 are the most useful, with values above 1 starting to make output look noticably choppy.

The last four textboxes along the right are x mult, y mult, z mult, and t mult. These are essentially scale values for the pitch (x), yaw (y), roll (z), and analog trigger (t) data. The values in the textbox are multiplied by the data coming from the device. For example, if the data from the device showed a rotation of 5 degrees in pitch (x) and the value in x mult was 2, the actual pitch value the software application sends to Maya would be 10, or 5 times 2.

The set target button (center, top row) specifies the data object in Maya to which we stream our data. In Maya, the user selects an object, and then hits the set target button. From then on, the rotational data and the analog trigger data affect that object. The most common object to be used for data streaming is a camera object. Other objects, such as locators, can also be used, albeit with slightly reduced functionality.

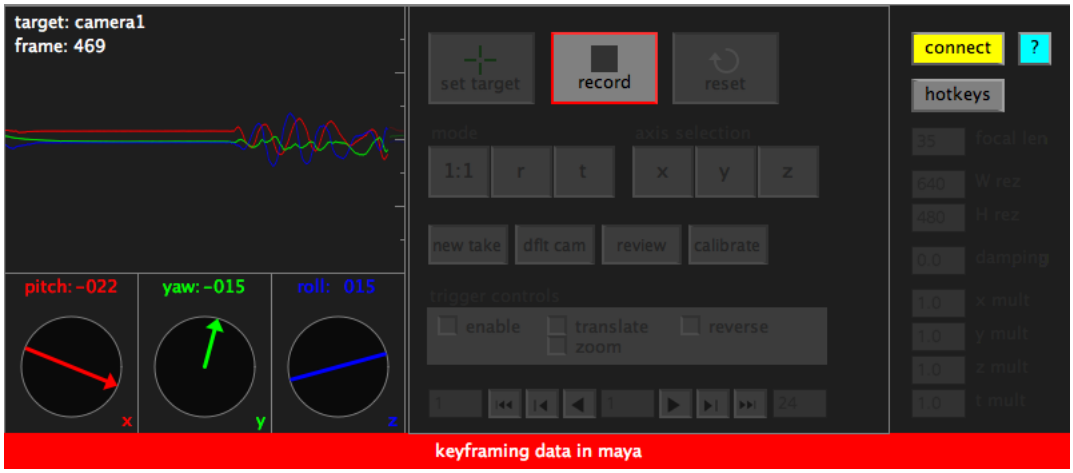

Figure 4.7: The software application in the midst of recording keyframes after the record button has been pressed.

The record button (center, top row) starts and stops keyframing of data. This button is not active until a target object has been set using the set target button.

When record is pressed, all other buttons in the application are disabled. Maya then sets a keyframe for the current target at the current frame, wait briefly, and then advance to the next frame. The frame advancement occurs at a rate equal to whatever the current framerate is set to. This process continues until the record button is pressed again, at which point frame advancement in Maya ceases and the other buttons are reactivated. At the end of a record, the current target has keyframed data of the rotations that took place.

The reset button (center, top row) clears any keyframes set on the target and sets the current frame to the beginning of the playback range. This functionality is very useful if the user is not satisfied with a motion capture performace by allowing them to quickly try again. Like the record button, this button is not active until a target object has been set.

Figure 4.8: The axis selection control, showing x and z axis as active and y axis as inactive.

The axis selection buttons (center, second row from top) control whether or not a given axis is "active." By default, all axes are active, which means they stream data to the target in Maya. The user can deactivate an axis by clicking the appropriate axis selection button. Data for that axis is still received from the device, but is not sent to Maya by the software application, nor is it keyframed if recording is initiated. Any combination of axes can be active at the same time.

The mode buttons (center, second row from top) give the user access to different control schemes. Only one mode can be active at a time. The default mode is 1:1 mode, where the rotation of the target in Maya directly mimics the rotations of the

device in the real world. In “r” mode, continuous rotation occurs in an axis where the speed of the rotation is proportional to the angle the device is tilted in that axis. So if we tilt the device 5 degrees about the x axis and hold it there, the target in Maya rotates at a slow, constant rate about the x axis. If we increase the tilt of our device to 20 degrees and hold it there, the target in Maya rotates at a faster constant rate about the x axis. If we tilt the device to 0 degrees and hold it there, rotation of the target ceases. The “t” mode is very similar to “r” mode. Continuous translation occurs down an axis where the speed of the translation is proportional to the angle the device is tilted. Unlike in “r” mode, the axis you tilt in does not correspond to the axis that the translation happens in. Originally this was the case, however upon testing this setup was far less logical than it was in “r” mode. Instead, in “t” mode, rotating in the y axis still moves the selected object in the y axis (up and down), but rotating about the x axis translates the selected object down the z axis, and vice versa. This corresponds more to more to a setup where translation occurs in the direction of the tilt itself, rather than in the axis about which the tilt occurs. Both the “r” mode and “t” mode tend to work best when only one or two axes are active, as having all three active at once can be very difficult to control.

The new take button (center, third row from top) creates a copy of your current target in Maya, hides the original, and then sets the target to the copy. This allows the user to quickly “save” a take they like while setting them up to try an additional take with the same settings. The default cam button creates a Maya camera object with default settings at the origin and sets the target to that object. This can be used as an alternative to the set target button, and provides the user with the quickest way to begin experimenting with the motion capture functionality. The review button plays through the frames in the current playback range once and then resets the current frame to the start of the playback range. This allows for a quick

review of the current keyframes. The calibrate button allows the user to reset the "zero point" or neutral state of the device. When the calibrate button is pressed, whatever orientation in space the physical device has is considered to have pitch, roll, and yaw of zero.

The trigger controls section of buttons (center, second row from bottom) provides options for utilizing the analog trigger. The enable button activates the trigger and allows its data to be sent to Maya. The user can then choose either translate or zoom. Choosing translate means a squeeze of the analog trigger affects the Translate-Z attribute of the current target. For a camera object, this means translation occurs in whichever direction the camera is facing. While the translate option works for any object, the zoom option works only for camera objects. Choosing zoom means a squeeze of the analog trigger changes the focal length of the target camera object, essentially zooming the camera. The final analog trigger option, reverse, either changes the direction of the translation (if translate is selected) or causes the focal length to get smaller i.e. more zoomed out (if zoom is selected). These controls combined with the t mult control already mentioned provide several options for the user when utilizing the analog trigger.

The playback controls (center, bottom row) for the most part mimic the functionality of the playback controls in Maya. The user can set the playback range (left and right textboxes) and the current frame (center textbox). The user can play forward or backward, advance forward or backward one frame, or skip to the first or last frame in the playback range.

The left third of the software application window does not provide any controls to the user, but rather helps represent what the current state of the user's data is. In the upper left, the current target and current frame are displayed. If no target has been set, NONE is displayed. Beneath that is a line graph that displays

approximately the last ten seconds of history of the pitch, roll, and yaw values. This is most useful for users that know how to read animation curves. The line graph continuously overwrites itself as it runs out of space. Finally, in the bottom left, we have both a numerical and visual display of the current pitch, roll, and yaw values. The numerical values represent the number of degrees the device is tilted in that axis, rounded to the nearest degree. Even though only whole numbers are displayed, decimal values are getting sent to Maya. Each circle gives a visual representation of one of the Euler angles as though you were viewing the device looking straight down the appropriate positive axis. The red line shows only pitch and represents looking at the device down the positive x axis (from the side), the green line shows only yaw and represents looking at the device down the positive y axis (from above), and the blue line shows only roll and represents looking at the device down the positive z axis (from behind).

4.2.2 Workflow

A typical workflow for using the software application begins with the user plugging the device into an available USB slot on their computer, and then launching the application. When the application first launches, it automatically recognizes the physical device, and it calibrates the device for approximately ten seconds. The user should hold the device in what they want to be the "neutral" or starting position during this time. The message center alerts the user when the calibration has been completed, and the left side of the application begins to update the various representations of the pitch, roll, and yaw with live values.

The next step is usually to connect to Maya, by first following the instructions to open a command port in Maya and then clicking the connect button. Once connected to Maya, the user selects the object they wish to stream data to in Maya and click

the set target button. At this point the user is able to rotate the physical device and see the results reflected as rotation of their selected target object in Maya.

When the user is ready to capture motion, they set their start frame with the playback controls and then click the record button. They move the device however they wish, then click the record button again. They can quickly review their captured motion by hitting the review button. If they do not like the result, they can hit the reset button to quickly try again using the same target object. If they like their result but would like to try another attempt, they can click the new take button to quickly try again with a new target object while preserving the keyframes they set on their initial object. The user can repeat this process until they have gotten the take they desire, using the functionality of the other buttons as needed.

5. RESULTS

5.1 Overview

The overall result of this research is a modular, extensible system which allows the user to generate rotational camera motion using physical movements, and where the physical movements have a direct relationship to the virtual camera motion produced. The system allows the user to manipulate the camera in a 3D software package in a dynamic and intuitive way.

The analysis of the results is organized into two sections. The first section covers the tests that were used to evaluate the system. The second section covers the strengths and weaknesses of the system as evidenced by the tests.

5.2 Tests

A series of basic tests were performed to ensure the various aspects of the system were working as expected.

The first test consisted of connecting the system to Maya and performing a series of rotations with the physical device while recording what was happening on-screen and in the real world at the same time. The two videos were then synced together in a single video for comparison purposes. The resulting video provided information regarding whether the system could indeed manipulate a virtual camera in Maya in real time. Also as part of the test, the analog trigger and all the controls in the software application were tested to ensure they were functioning as intended. A screenshot from part of this test can be seen in Figure 5.1.

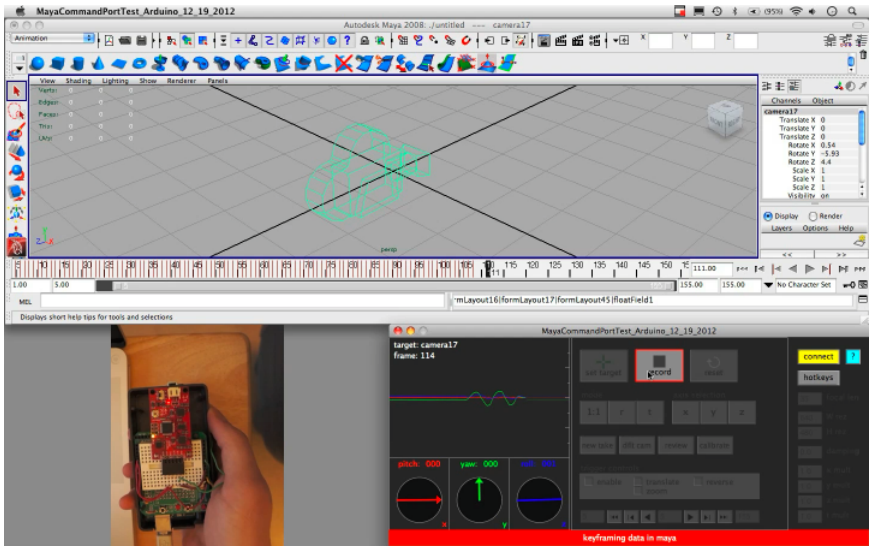

Figure 5.1: A screenshot from the video of the first test.

This first test served as a qualitative accessment of how well the system functioned. Using our system is an interactive manual process where the user is looking at real-time feedback on a screen to aim the camera. In such a situation, the operator or cameraperson naturally compensates for small inaccuracies between the physical device and the virtual camera, as long as the inaccuracies never become too significant. The results of our first test showed that our system was always "close enough" such that any inaccuracies did not affect the natural operating procedure of the system. In other words, the system was never so far off that a rotation performed in the real world with the physical device did not appear to match up with what was happening on-screen to the virtual camera.

This test gave us additional valuable insight. It provided the first empirical evidence that the system was functioning as intended at a basic level. It confirmed that our system could send data from the physical device through the software application into Autodesk Maya and finally to the rotation attributes of a virtual camera in Maya. Upon observation, it was clear that the data sent had a direct correlation

to the movement of the physical device. This test also gave us the first opportunity to establish and test a workflow using the controls provided in the software application. Iterations were fast and easy. This test also confirmed the analog trigger was functioning as intended, both in zoom and translate mode.

The second test was designed to determine how camera motion generated by the device compared to camera motion generated by a real camera. For this test, the physical device was mounted to the top of a Sony NEX-5N camera that is capable of video capture. The camera with the physical device attached was then mounted to a tripod and oriented to point towards a test scene. Figure 5.2 below shows the testing setup.

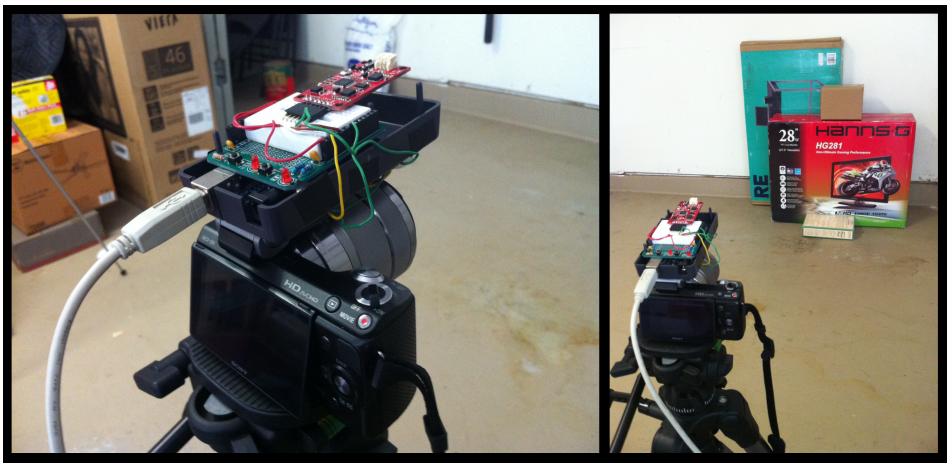

Figure 5.2: The setup used to compare motion captured by the system in this thesis to real-life camera motion.

Using the tripod, the camera was put through motions to show pitch, roll, yaw, and a combination of all three. While these motions were being performed, the physical device was recording motion data. The layout of the test scene was logged, including the dimensions of the set pieces, the height of the tripod, the distance of the camera from the scene, and the focal length of the lens. The test scene was then reproduced in Autodesk Maya as accurately as possible to the original real

world scene. The motion data from the camera tests was then applied to a series of virtual cameras in Maya, and the virtual test scene for each test was rendered out. Finally, the videos of the CG test scene using camera motion captured by the physical device were synced to and compared to the videos of the real test scene. A series of screenshots from one these videos can be seen in the Figure 5.3 below.

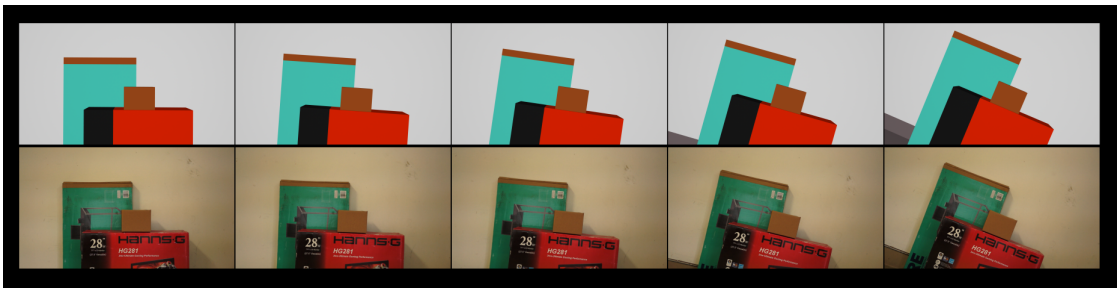

Figure 5.3: A series of screenshots from a test video comparing roll captured with the system from this thesis (top row) to real-life camera motion (bottom row).

To further compare the CG and live action footage, the two synced clips were superimposed on top of each other at reduced opacity. A series of screenshots from this setup can be seen in Figure 5.4. While the CG footage was still comparable to the live action footage, the differences between the two were also much more apparent. It is important to note that a perfect 1:1 correspondence between the two sets of footage was unfeasible, as there were many extenuating factors that resulted in discrepancies between the two sets of footage. The live action scene was measured as accurately as possible, but the CG scene was most likely still not a completely accurate representation of the live action scene. The live action footage was filmed using a lens that has known barrel distortion (which visibly bends objects towards the side of the frame), while the virtual camera obviously did not suffer from this barrel distortion. Even though the camera was mounted on a tripod, the pivot point was not perfectly centered. This meant that the camera was translating slightly with every movement, which could not be picked up by our system nor measured

accurately enough to compensate for. As a result of all of these inconsistencies, the CG and live action did not match up exactly.

Figure 5.4: The same two sets of screenshots from Figure 5.3 are now superimposed on top of one another at reduced opacity.

The second series of tests was much more rigorous than the first test. It provided an absolute comparison between what rotations the system detected and what rotations were actually occurring in the real world. Viewing the two sets of synced footage side-by-side and then superimposed on top of one another removed any compensation for inaccuracies the human brain might be doing. Even in this more direct comparison, the data produced by our system held its own, showing no functionally significant differences when compared to the live action footage. It always maintained a similar orientation to the live action footage and never fell behind in the timing of movements. It maintained its accuracy well, with the ending position of the CG scene always closely mimicing the ending position of live action scene, even after multiple rotations. In the end, the side-by-side and superimposed comparisons together provide enough evidence to conclude that our system can produce virtual camera motion that is directly comparable to real camera motion and sufficiently similar enough to be useful.

5.3 Strengths and Weaknesses

Overall, the system performs quite well. First and foremost, the device works as intended and has the potential to be genuinely useful. The physical device, along with the software application, can detect and report pitch, roll, and yaw in real time and can be connected to Maya with very little difficulty. When compared to live action

footage shot under the same circumstances, the virtual footage captured with this system is objectively similar. The system is good at maintaining all the "zero points" on each axis. In other words, if it is rotated one way and then back to its original starting location, it reports that it is indeed back in the same starting position, and not off by more than a fraction of a degree. The system can maintain this accuracy even after extended use (the longest tests performed were several minutes in length, which is significantly longer than an average shot in a movie) and multiple rotations. Data noise is well controlled, and the user has the option to dampen the noise even further at the expense of precision and responsiveness. The software application offers myriad options to the user, and facilitates a quick, iterative workflow.

The ease of expansion of the physical device is a great advantage of the system. To add the analog trigger to the system, very little physical electronics work was needed, and only a few additional lines of code were required. The additional physical control was added without having to change the core code written for motion capture. Thanks to the Arduino board, the system can accept several more controls if need be, which allows for extensive customization options.

The system does have a few weaknesses. The more extreme angles of rotation (around 90 degrees in the pitch and roll and 180 degrees in the yaw) tend to be less precise when measuring angle changes than when the angles are less extreme. These types of angles tend to be rare in real cinematography (i.e. most camera operators or camera rigs cannot even physically roll a full 90 degrees), but it is still a weakness of the system. There is a small amount of lag between moving the device and the motion registering on-screen in Maya. In our tests the amount of lag was not significant enough to cause problems. In other words, the lag never reached a point where it hindered or prevented us from making timed reactions to what was occurring on-screen. Even so, the amount of lag present could potentially be an

issue in certain situations. If rotation is too fast (over 300 degrees per second) for too long (1 to 2 seconds), some of the sensors actually get overloaded and are not able to report accurate data. Rotations of this speed typically only occur with very fast camera movements like whip pans. Most other camera movements of that speed and duration would result in footage so fast and blurry that it would be almost unusable. Finally, this system was designed to work using a USB cord to connect to a computer. While USB cords come in a variety of lengths, being tethered to a computer does not allow for the same freedom of motion or flexibility that a wireless connection would provide.

6. CONCLUSION

The primary goal of this research was to develop an affordable, extensible system that allowed a user to capture rotational camera motion quickly using physical motions that mimic the desired camera movements.

This thesis focused on creating a prototype system composed of a physical device and a software application that work together to produce rotational motion data.

The system developed successfully captures pitch, roll, and yaw motions that very closely mimic actual motions performed by the user. Using freely available software and off-the-shelf commercially available hardware means our system is affordable enough to be built and programmed by an independent individual, and it allows for very quick and easy expansion of the physical controls. The workflow provided by the device and software allows for quick iterations, and the physical device facilitates a what-you-see-is-what-you-get control scheme that traditional mouse-and-keyboard techniques cannot.

The system is not without its weaknesses, and in certain situations more traditional methods for creating virtual camera motion would be appropriate. In addition, though it was possible to generate realistic virtual camera motion with the tools and processes offered by this thesis in our tests, we must concede that they are prototypes, and have not been fully tested as artistic tools in an actual production environment. The quality of this system cannot compare with larger, more expensive systems developed by professional companies.

Nevertheless, the results of our research clearly show potential for this system. For smaller productions, it has the potential to be used to effectively and quickly to achieve the user's artistic goals in many circumstances, and it can be improved and

expanded upon to increase its functionality as the situation may require.

7. FUTURE WORK

The system we have developed in this thesis opens the door to various different projects, including opportunities to develop new tools and systems to aid the user in their artistic goals.

The current system is designed only to work with Autodesk Maya. With a few changes, it could be adapted to work with other 3D software packages such as 3ds-Max, Cinema4D, Blender, or even proprietary software. The software application is currently coded in Processing, but could be updated to be used as a plugin for different 3D software packages. A redesign of the software application's user interface that follows better UI and UX design principles has the potential to streamline the workflow of the system.

Further developing the capabilities of the current system could result in additional value. As previously mentioned, this research did not focus on providing a true camera-like user experience. Mounting the system to a rig that emulates a real camera's weight and feel, while also providing standard on-camera controls like a zoom, record button, and a viewfinder has the potential to offer an more natural and intuitive experience than our current system. Adding wireless capabilities via a method such as bluetooth would for less restricted movement. The system could also potentially be modified to write to a memory card or other storage device instead of connecting directly to Maya, which would allow the system to be used without being connected to a computer.

The current rotational motion sensing capabilities of this system could be adapted for other uses in the visualzation field. Visual effects professionals often have to insert computer-generated imagery (CGI) into a real-world footage. If the real world

footage moves or changes perspective in a shot, the user must use a process called matchmoving to insure that the CGI moves properly with respect to the real-world footage. Adapting our system to be mounted on top of a film camera and record to a memory card would allow the system to capture rotational data while the film camera captures footage. The captured rotation data could then be accessed during the matchmoving process to give the user a potentially better starting point than if he had not had rotational data. Rotational data from this system could also be used for homebrew head tracking in virtual reality or immersive visualization setups by using the motion data to adjust the viewing perspective based on how the user is moving their head.

One obvious future project would be to create a system that captures both rotational and translational motion data for virtual camera control. With rotational motion capture already developed, the future work would focus on using our exisiting system in the context of another system to capture translational motion. In particular, we believe the Kinect sensor and its newer successor the Kinect 2 , both developed by Microsoft, have great potential. The Kinect family of sensors uses an infrared projector and camera and a special microchip to track movement of objects. In tech demos Microsoft has demonstrated that at a distance of approximately six feet away, the Kinect 2 can see the entire human body from head to toe, and can detect specific hand movements and gestures as well as changes as subtle as whether or not a user's eyes are open or whether they are smiling or frowning [25]. This level of accuracy and fidelity leads us to believe that, with a bit of programming, the Kinect 2 can potentially perform object tracking that is accurate enough to be usable for virtual camera motion. The Kinect sensor and a software development kit for Windows were released during the implementation of this thesis, and several interesting projects that utilize it have already been developed.

REFERENCES

[1] Accelerometer, Gyro and IMU Buying Guide. Retrieved July 14, 2012: http://www.sparkfun.com/pages/accel_gyro_guide.

[2] Coriolis effect. Encyclopedia Britannica. Retrieved August 5, 2013: http://abyss.uoregon.edu/ js/glossary/coriolis_effect.html.

[3] Creative commons attribution-sharealike 3.0 unported. Retrieved September 17, 2013: http://creativecommons.org/licenses/by-sa/3.0/.

[4] Game Controller Triggers (left/right pair). Retrieved October 10, 2013: https://www.sparkfun.com/products/10314.

[5] Performance captured. www.creativeplanetnetwork.com. Retrieved Jan 3, 2014: http://www.creativeplanetnetwork.com/dcp/news/performance-captured/44694.

[6] Sparkfun — 9 degrees of freedom — razor imu — ahrs compatible. Retrieved January 4, 2014: http://robosavvy.com/store/product_info.php/products_id/625.

[7] Adxl345 datasheet. Analog Devices, 2009. Retrieved February 9, 2012: http://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf.

[8] A guide to using imu (accelerometer and gyroscope devices) in embedded applications, December 2009. Retrieved April 11, 2013: http://www.starlino.com/imu_guide.html.

[9] Hmc5883l compass tutorial with arduino library, July 2011. Retrieved October 15, 2012: https://www.loveelectronics.co.uk/Tutorials/8/hmc5883l-tutorial-and-arduino-library.

[10] Arduino — wikipedia, the free encyclopedia, 2013. Retrieved October 12, 2013: http://en.wikipedia.org/wiki/Arduino.

[11] Arduino - homepage, 2013. Retrieved October 12, 2013: http://www.arduino.cc.

[12] A beginner's guide to accelerometers, 2013. Retrieved August 3, 2013: http://www.dimensionengineering.com/info/accelerometers.

[13] Coriolis effect — wikipedia, the free encyclopedia, 2013. Retrieved October 23, 2013: http://en.wikipedia.org/wiki/Coriolis_effect.

[14] Data transmission — wikipedia, the free encyclopedia, 2013. Retrieved August 12, 2013: http://en.wikipedia.org/wiki/Data_Transmission.

[15] Digital electronics basics - chapter 3: Registers, March 2013. Retrieved September 6, 2013: http://www.ni.com/white-paper/14495/en/.

[16] Image — compass.jpg, July 2013. Retrieved October 5, 2013: http://pikmin.wikia.com/wiki/File:Compass.jpg.

[17] Tilt compensating a compass with an accelerometer, 2013. Retrieved November 9, 2012: http://www.loveelectronics.co.uk/Tutorials/13/tilt-compensated-compass-arduino-tutorial.

[18] Wii — wikipedia, the free encyclopedia, 2013. Retrieved October 24, 2013: http://en.wikipedia.org/wiki/Wii.

[19] Wii remote — wikipedia, the free encyclopedia, 2013. Retrieved October 24, 2013: http://en.wikipedia.org/wiki/Wii_Remote.

[20] Baker, M. Physics - kinematics - combined linear and angular velocity, 1998. Retrieved September 3, 2013: http://www.euclideanspace.com/physics/kinematics/combinedVelocity/.

[21] Brannon, A., and Buck, C. Surf's up. Sony Pictures Animation, 2007.

[22] Desowitz, B. Hello, WALL-E!: Pixar Reaches for the Stars. *Animation World Magazine June* (2008).

[23] Erbland, K. Robert zemeckis just really into flight-based films now; will develop 'taking flight' next, August 2012. Retrieved October 4, 2013: http://filmschoolrejects.com/news/robert-zemeckis-taking-flight-kerbl.php.

[24] Esfandyari, J., Nuccio, R. D., and Xu, G. Introduction to mems gyroscopes. Solid State Technology, November 2010. Retrieved June 11, 2012: http://www.electroiq.com/articles/stm/2010/11/introduction-to-mems-gyroscopes.html.

[25] Goldfarb, A. How next-gen kinect is more accurate than ever, November 2013. Retrieved Feb 5, 2014: http://www.ign.com/articles/2013/11/07/how-next-gen-kinect-is-more-accurate-than-ever.

[26] Holben, J. Conquering new worlds, January 2010. Retrieved Jan 17, 2014: http://www.theasc.com/ac_magazine/January2010/Avatar/page1.php.

[27] Joshi, A. How does a mems gyroscope work? Quora, April 2012. Retrieved June 6, 2012: http://www.quora.com/How-does-a-MEMS-gyroscope-work#.

[28] Kabal, S. Gyroscope. Wolfram Demonstration Project, September 2007. Retrieved September 1, 2013: http://demonstrations.wolfram.com/Gyroscope/.

[29] Kochanek, D. H. U., and Bartels, R. H. Interpolating splines with local tension, continuity, and bias control. In *SIGGRAPH '84 Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1984), SIGGRAPH '84, ACM, pp. 33–41.

[30] Kolb, C., Mitchell, D., and Hanrahan, P. A realistic camera model for computer graphics. In *SIGGRAPH '95 Proceedings of the 22nd annual confer-*

ence on Computer graphics and interactive techniques (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 317–324.

[31] Potmesil, M., and Chakravarty, I. A lens and aperture camera model for synthetic image generation. In *SIGGRAPH '81 Proceedings of the 8th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1981), SIGGRAPH '81, ACM, pp. 297–305.

[32] Riesterer, J. Introduction to topographic maps: Magnetic declination. Geospatial Training and Analysis Cooperative, April 2008. Retrieved November 9, 2012: http://geology.isu.edu/geostac/Field_Exercise/topomaps/magnetic_declination.htm.

[33] Robertson, B. Shooting animation verita-style for surf's up, 2007. Retrieved March 17, 2013: http://www.studiodaily.com/2007/06/shooting-animation-verita%C6%92a-style-for-surfs-up/.

[34] Rose, F. The creation, January 2010. Retrieved October 4, 2013: http://www.frankrose.com/avatar-the-creation.html.

[35] Shoemake, K. Animating rotation with quaternion curves. In *SIGGRAPH '85 Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1985), SIGGRAPH '85, ACM, pp. 245–254.

[36] Stanton, A. The imperfect lens — creating the look of wall-e. Disney/Pixar, 2008.

[37] Stern, D. P. Accelerated frames of reference: Inertial forces, 2004. Retrieved October 14, 2013: http://www-istp.gsfc.nasa.gov/stargaze/Sframes2.htm.

[38] Terdiman, D. Why sony imageworks got an oscar nod for 'surf's up', 2008. Retrieved March 17, 2013: http://news.cnet.com/8301-13772_3-9862945-52.html.

[39] Turner, R., Balaguer, F., Gobbetti, E., and Terdiman, D. Physically-based interactive camera motion control using 3d input devices. *Scientific visualization of physical phenomena* (1991), 135–145.

[40] Ware, C., and Osborne, S. Exploration and virtual camera control in virtual three dimensional environments. In *I3D '90 Proceedings of the 1990 symposium on Interactive 3D graphics* (New York, NY, USA, 1990), I3D '90, ACM, pp. 175–183.

[41] Weisstein, E. W. Vector Triple Product. MathWorld–A Wolfram Web Resource. Retrieved September 27, 2013: http://mathworld.wolfram.com/VectorTripleProduct.htm.

[42] Wise, S. The scientific tourist 29 — the foucault pendulum, July 2008. Retrieved May 28, 2013: http://sortingoutscience.net/2008/07/15/the-scientific-tourist-29-the-foucault-pendulum/.