Smartphone Detection of Abnormal Equine Behavior

by

Megan Mariah Burton

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
December 15, 2018

Approved by

Dr. David Umphress, Chair, Professor of Computer Science and Software Engineering
Dr. James Cross, Professor of Computer Science and Software Engineering
Dr. Dean Hendrix, Associate Professor of Computer Science and Software Engineering
Dr. Robert Lord, Professor of Systems Engineering, Defense Acquisition University

Abstract

This research was motivated by two key facts: There are over 9.2 million horses in the United States and over 64% of adults own a smartphone. Blending these two ideas led to the genesis of this research by asking the question: "Can an unmodified, off-the-shelf smartphone be used to detect and categorize behavior of an equine in a controlled setting?" This research used computer vision techniques and extended game-based modeling to describe patterns of behavior that are considered normal, to determine when observed behavior falls outside those patterns, and to diagnose the possible cause of the anomaly. The research resulted in a proof-of-feasibility system that demonstrated use of a smartphone to differentiate normal behavior from abnormal behavior – pawing, in this case – of an equine while in a stall.

Acknowledgments

Working toward my PhD has been one of the most challenging yet rewarding endeavors I have embarked upon. There have been so many people who have supported and encouraged me along the way, and I deeply appreciate each and every one of you.

I would like to thank my advisor, Dr. David Umphress, for his encouragement, feedback, and guidance along every step of this process. Thanks to my committee, Dr. James Cross, Dr. Dean Hendrix, Dr. Robert Lord, and my University Reader Dr. Matt Miller. Each sacrificed their time to review this work and provide feedback, and I really appreciate it. Also a special thank you goes to Dr. Jennifer Taintor for providing her subject matter expertise.

I'm also forever grateful to my parents and grandparents for their unfailing love, support, and for instilling in me a lifelong love of learning. I love you all! Finally, and most importantly, my love and appreciation go to my wonderful husband, Scott Burton. Thank you for putting up with a messy house and many nights of cereal for dinner while I was busy studying. I love you with all my heart!

Table of Contents

List of Tables

List of Figures

CHAPTER ONE: INTRODUCTION

NEED FOR THE RESEARCH

Having an automated way for equine[1] owners to be quickly notified of their equine exhibiting abnormal behavior in their equine affords the opportunity for early intervention in potentially life-threatening situations.

"There are 9.2 million horses in the United States" (American Horse Council, 2005, p. 1). Although the per-year weighted average of veterinary services per equine is $251, emergencies can quickly run into the thousands of dollars (American Horse Council, 2005). Quick detection of anomalous behavior of an equine can lead to catching problems early while they can still be corrected quickly and inexpensively.

Smartphones are prevalent in the United States with "64% of American adults" owing a smartphone (Smith, 2015, para. 5). The idea of this research was to combine these two pieces of information by developing a simple, easy-to-use system to detect abnormal behavior in equines using something most equine owners have -- a smartphone. This system allows the average equine owner to monitor equines behavior without specialized expensive equipment – just common self-adhering bandages.

[1] For the purpose of this dissertation the broader term equine is used except when horse occurs in direct quotes. The term equine encompasses both horses and mules.

Equines can exhibit a wide variety of abnormal behaviors in a stall: pawing, cribbing, weaving, pacing, rearing and striking, just to name a few. This research focused on detecting pawing and distinguishing it from normal stall behaviors such as walking and standing.

Pawing can be a sign of stress or pain. It can also be a sign of colic which is a collection of symptoms that alert the equine owner to abdominal pain in an equine (Butler & Houpt, 2014). Colic can range from mild to severe, and can rapidly turn fatal (UC Davis Center for Equine Health, 2008). Colic is "the most common cause of death in adult horses and accounts for a large proportion of emergencies for horse owners and veterinarians" (UC Davis Center for Equine Health, 2008, p. 1). If caught early enough, medication alone can often halt the progress of colic. If not caught early, surgery is frequently the only way to save the equine. The majority of equine colic surgeries range from $3500 - $5,500 (Colorado State University, 2014).

BASIC APPROACH

Using the onboard camera of a smartphone, this research focused on detecting the differences between normal and abnormal behavioral patterns by using the onboard camera of a smartphone to track behavioral markers. This research focused on the behavior exhibited by an equine in a stall. The research and development were divided into three phases: modeling, data acquisition/behavior recognition, and analysis.

Modeling Phase

During the modeling phase, video of an equine in a stall was taken using a smartphone mounted in the equine's stall. This video was used to determine the equine's behavioral patterns while in

a stall. These behavioral patterns were further synthesized into behaviors markers that were encoded into a software behavioral model.

Data Acquisition/Behavior Recognition Phase

During this phase, the smartphone captured images of the equine at short discrete intervals, isolated behavioral markers, postulated observed behavior based on a best match to modeled behaviors, and reported the results in near real-time. A second smartphone was placed in the stall next to the first smartphone and continuously recorded the equine's behavior.

Analysis Phase

During the analysis phase, each frame of the video recording was compared to the smartphone's output to ensure behavior was properly identified.

CHAPTER TWO: LITERATURE REVIEW

OVERVIEW

The long-range aim of this research was to employ low-cost consumer electronic products to detect and diagnose equine abnormalities. The immediate term goal was to use a commercial off-the-shelf smartphone to determine if an equine in a confined area -- a stall, in this case -- is behaving abnormally and to diagnose the possible cause. The research elements of this work are:

1) Model the equine in such a way that it can be represented in a logical form suitable for computation.
2) Acquire data for the model in real time from an actual animal using sensors available on a smartphone.
3) Recognize abnormal behavior and diagnose the possible cause.

Figure 2.1 illustrates the relationship among these research elements.

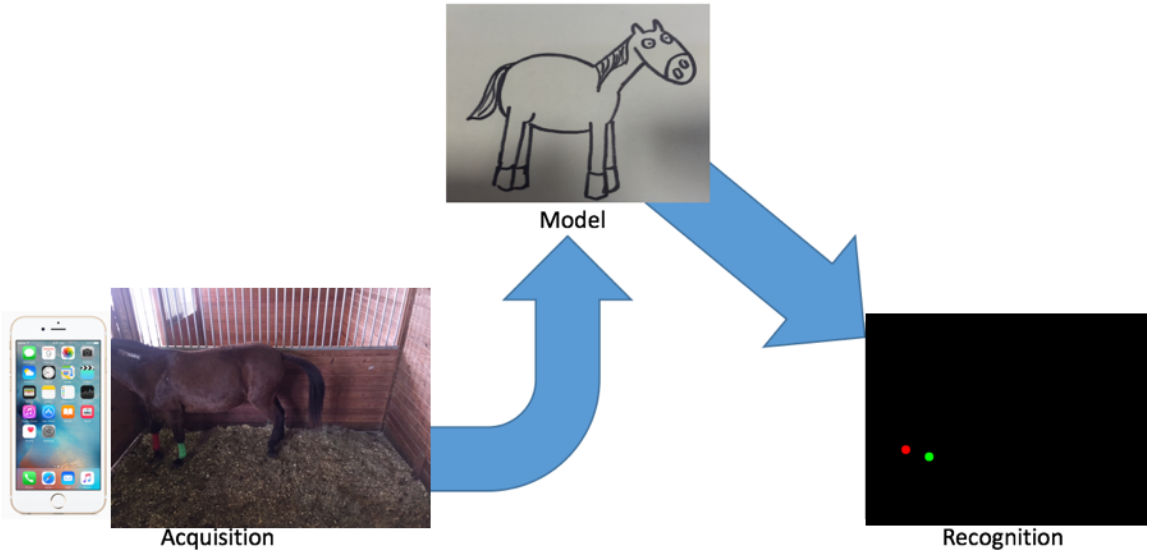

Figure 2.1: Relationships Among Research Elements

MODEL

A model is an abstract representation of an object being observed. It embodies information at a level of specificity that was sufficient to describe, in this case, what constitutes abnormal equine behavior, while excluding details that were deemed extraneous.

Modeling an equine at the subatomic particle level would provide the greatest fidelity to detecting abnormalities, but doing so would require a level of granularity that is not computationally feasible, much less fully understood. The modeling process is thus a balance of determining what equine properties were needed to identify abnormal behavior, what properties could be captured by a sensing device, how to represent those properties, and so forth.
Three primary means are used to model living beings: kinematic, shape, and appearance. Kinematics expresses the geometry of motion, typically in terms of joints and their respective degrees of freedom (Beatty, 1986). The most important areas of kinematic research are the number of joints and degrees of freedom the subject under study has. With a kinematic model, the goal is to find and estimate the limb lengths of the subject (Moeslund et al., 2006). The papers surveyed in (Moeslund & Granum, 2001; Moeslund et al., 2006) deal with a human model, which does not directly translate to this dissertation's area of research since the number of joints and degrees of freedom in a human is vastly different than those in other animals.

Shape models represent objects with various shapes that can be either simple (such as cylinders and cones) or complex (such as a polygonal mesh). An important consideration of a shape model is how much fidelity is needed in the model to ascertain the behavior being examined. For example, representing the horse as a collection of cylinders and cones would be appropriate

to describe large-muscle movements such as pawing but might not have enough fine-grain detail to capture the blinking of an eye. Computational power also limits the complexity of the shape model. Detailed shapes demand more computational resources than do simple shapes. Regardless of the complexity of the shape chosen, the goal in shape modeling is to fit a generic model to the actual subject in the image (Moeslund & Granum, 2001).

Appearance models make certain assumptions about the images being examined. The assumptions can be related to the subject (known starting pose, markers placed on the subject, special colored clothes, etc.) or the environment (uniform background, constant lighting, static background, etc.) (Moeslund & Granum, 2001). An example of using an appearance model with an equine in a stall is to make the assumption about what color belongs to the horse and what belongs to the stall. One difficulty of appearance modeling is choosing something the equine will display that the background will not. For example, bright red is seldom the color of a stall or a color that is natural to equines, making it feasible to assume that it is some sort of purposely-placed positional marker if detected. This assumption may not be generalizable to all settings. Having a water bucket in the stall that is close to the same bright red color would cause the appearance model to mistake the bucket as being part of the equine.

SENSOR ACQUISITION

Model acquisition populates a model with information about a specific subject (in this research the specific subject was an equine). It entails collecting information about the subject using sensors and then interpreting the collected sensor information into the model. There are two broad categories of sensors: optical and non-optical.

Non-Optical Sensors

Non-optical sensors include any sensors that do not detect wavelengths of light and "include magnetic sensors, inertial sensors, pressure sensors, and temperature sensors, among others" (Persistence, 2015, para. 1).

An example of a non-optical sensor technique is electromagnetic tracking. The Polhemus' Fasttrack electromagnetic tracking system was used to study the kinematics of snowboarder's ankles (Delorme et al., 2002; Polhemus, 2016). This particular technology appears to be sensitive to the environment in which measurements are taken, especially if tracking is done near large metal objects (Dutta, 2012). While there is an indication that only certain metals affect accuracy, the efficacy of electromagnetic tracking seems marginal in an equine environment that includes a metal barn (Milne et al., 1996).

An application of non-optical sensors to equine research is the prototype MVN Equine suit developed by Xsens. "The system utilizes inertial sensors located on the horse's body and GPS to track full-body motion in any environment, indoors and outdoors, allowing the horse's innate, voluntary movements to be recorded and viewed on a standard PC in real-time" (Xsens, 2016, para. 2).

A force plate is a type of non-optical sensor used with horses to measure how the horse distributes weight across its hoof. Penn works on a system that uses a force place to help farriers in the future "develop orthotics that help horses distribute weight more evenly across their feet" (Penn, 2012, para. 19). This technology could be used to detect pawing in equines as the pawing hoof would be hitting the surface with a greater force than the other non-pawing hooves.

An active area of research with non-optical sensors and smartphones involve using the data from built-in smartphone sensors (such as the accelerometer) to recognize the activity of the human user (Anguita et al., 2012; Dernbach et al, 2012; He & Li, 2013; Khan et al., 2010; Ouchi & Doi, 2012). However, the research literature shows no application of smartphones for tracking the movement of equines. A comparable approach pursued by researchers is to attach sensors to the human and feed the data into an application on a smartphone or a specialized computer (Győrbíró et al., 2009). This approach is used with equines for heart rate monitoring during training. With a current system available from Polar, sensors are attached to the equine and the equine's heart rate is displayed on a specialized computer worn by the rider on the wrist (Polar, 2016). Such a system could be useful for behavior monitoring in equines since heart rate increases when an equine is in pain or stressed.

Optical Sensors

An optical sensor is a sensor that measures the wavelength of light. Optical sensors specifically measure wavelengths found in the visible, ultraviolet, or infrared regions of the electromagnetic spectrum (Melexis, 2016). Optical systems can be divided into markerless and marker-based systems. Marker-based systems can be further divided into those that use active markers versus those that use passive markers. Active markers are markers that emit light while passive markers only reflect light (Barca et al., 2006; Endgadget, 2016; Qualisys, 2016). Marker-based systems are the most popular systems for capturing human motion (Corazza et al., 2006).

Depending on the application, markerless and marker-based approaches can provide similar results. For example, Rosenhahn et al. (2006) compare their markerless system with a

commercially available marker-based system. Their research found the root mean square (RMS) errors between the marker-based approach and the markerless approach were typically within three degrees of each other (Rosenhahn et al., 2006).

Similarly active markers and passive markers can give comparable results (again, depending on the application). Wiles et al. (2004) discuss the differences between surgical tools tracked with active markers versus surgical tools tracked with passive markers. The authors discuss a common misconception that tools equipped with active markers are more accurate than tools equipped with passive markers. However, the researchers found for the brand of sensors they used in the study "tools equipped with passive markers can be tracked as accurately as similar tools equipped with active markers, contrary to beliefs held by many users" (Wiles, et al., 2004, p. 427).

Markerless Systems

As the name implies markerless systems do not use any sort of marker on the subject, relying, instead on a kinematic or shape-based model of the subject to initialize the system (De Aguiar et al., 2007). Markerless systems require more computational steps than marker-based approaches because in marker-based approaches the markers are identified externally from the algorithm but in markerless systems the algorithm must first determine what are good markers of the object to track. As Bregler (2007) states, it is "more difficult to track pixels with arbitrary surface texture than with retro-reflective balls" [i.e. markers](Bregler, 2007, p. 156).

There have been a variety of algorithms proposed for markerless systems. The algorithms can be placed into two general categories: 1) Algorithms which involve statistical training based on

local features and 2) Algorithms which involve extracting interest points in the image. Xia et al. (2011) reports markerless methods can provide accurate results when detecting a human in a scene but have limitations especially when the background is cluttered. Accuracy drops and the computational intensity increases as the background becomes more cluttered (especially with objects similarly colored to the human) (Xia et al., 2011).

Usually, markerless systems use two or more synchronized cameras to capture the motion of the subject. However, researchers like Okada & Stenger (2008) have been investigating how to apply markerless techniques to single camera images. Researchers at Disney developed a process to estimate human motion using only video captured from a single video camera. This process is computationally intensive with optimization of 200 frames taking 2.75 hours (Vondrak et al., 2012).

There are several commercial systems which provide markerless motion capture. For example, Organic Motion's OpenStage motion capture system has "8 cameras to 24 cameras and can cover areas ranging from four-square feet to 30-square feet". (Takahashi, 2011, para. 4). The starting price of $40,000 places such systems out of the reach of the consumer electronic market.

ProAnalyst, another commercial product, is a software tool that has been used by researchers to analyze the gait of a racehorse. A video of a racehorse is taken and the ProAnalyst software performs markerless feature tracking to measure the racehorse's strides from a video of a racehorse. Although less expensive than the OpenStage capture systems, ProAnalyst is still beyond the cost of ordinary consumer electronics. The introductory software is $1,795 while the 3-D professional version is $13,195. The base model of hardware is $2,795 while the most

expensive is $8,295. There are also various add-on software packages at roughly $1,000 a piece (Xcitex, 2016).

The computational requirements of markerless systems have placed them outside the capabilities of smartphones. We found no instances in which smartphones performed markerless sensing. The closest was research performed by Quesada & León (2011) which used an external USB webcam, a laptop integrated webcam, or a HD camcorder to perform markerless optical motion tracking. They did not discuss what hardware was running their algorithms, but pointed out that "We plan to optimize the system so that it can run in low processing power devices such as smartphones" (Quesada & León, 2011, p. 10). In subsequent work, they replaced the ray-casting technique in the original research with a grid-filling technique in order to reduce computational requirements for low-budget hardware (Quesada & León, 2012). Quesada & León (2012) presents no further evidence of the system working on a smartphone.

Although smartphones are not currently conducive to markerless techniques, one system that has gained a lot of attention lately in markerless research is the Microsoft Kinect. The Kinect combines "depth sensing technology, built-in color camera, infrared (IR) emitter, and microphone array" into a single system (Kinetisense, 2018), para. 5).

Unlike the traditional markerless based approaches discussed above that only use visible-light cameras, the Kinect adds the important element of depth which humans (and most animals) use to help distinguish objects. Cameras that add this depth information are referred to as RGB-D cameras

Research with the Kinect sensor has been done in fields ranging from gaming to health care to robotics. One use of the Kinect presented by Rocca et al. (2016) is to identify when the user is

watching TV versus when the user is not paying attention and instead staring at a secondary device like a tablet or smartphone (Rocca et al., 2016). In the healthcare industry, the Kinect is being evaluated for its use in physical therapy. Garrido et al. (2013) used the Kinect sensor to develop a system for patients with a balance disorder to perform rehabilitation exercises at home instead of having to stay at a rehabilitation facility.

The Kinect is an array of sensors and does not have onboard processing. It either must be attached to a PC running Windows or be used as part of Microsoft's gaming system, XBOX. Additionally, RGB-D cameras (such as the Kinect) have the following operational limitations: "1. Limited field of view preventing an agile operation. 2. Short range, not providing the scale for typical outdoor applications. 3. Infrared saturation in direct sunlight" (Abbas & Muhammad, 2012, p. 1).

There has been some published research on use of the Kinect with animals. Stavrakakis et al. (2015) uses the Kinect sensor to detect the normal walking pattern of pigs. They compared a marker-based Vicon system (consider the gold standard) to a marker-based Kinect system to a markerless Kinect system. Both marker-based systems outperformed the markerless system and showed a "high level of agreement" with each other (Stavrakakis et al., 2015, p. 6). They stated the following about the markerless approach: "Thus fully automated and marker-free tracking of relevant dorsal mid-line point trajectories for a relatively modest cost appears to be feasible, but the technology requires refinement and further software development before it can be recommended for commercial use" (Stavrakakis et al., 2015, p. 6). With a similar highly controlled set-up, such an approach could be possibly be applied to equines as well.

One use of an RGB-D camera with equines was done by Ku (2014). He used a Structure Sensor, which is an RGB-D sensor similar to the Kinect, to perform a 3D scan of an equine (Structure,

2016) (Ku, 2014). The data obtained from this 3D scan was used to estimate the body condition of the equine (Ku, 2014).

Marker-Based Systems: Active

Because active systems rely on markers that emit light, they require the device to have a power source (such as a battery) or to emit light through a chemical reaction. There are a variety of different types of active markers as demonstrated through the selection of papers below.

Barca et al. (2006) developed a new active marker system that "supports active tracking through the use of a battery- driven low power color marker and is ideal for tracking non-rigid motion" (Barca et al., 2006, p. 2). Kumar et al. (2010) uses light-emitting diodes (LEDs) as active markers to perform kinematic analysis of human's gait. Conceivably such a system could also be used to monitor equine's gaits.

Smyth et al. (2015) used an active marker-based system to study whether injuries to the temporomandibular joint (TMJ) affect how equines eat hay. "Kinematics of the head and mandible were recorded using an active- marker motion capture system (Visualeyez VZ3000) a that tracks the 3-dimensional (3D) locations of small (~5 mm diameter) infrared emitting markers (nominal resolution of 0.3 mm)" (Smyth et al., 2015, p. 1). Figure 2.2 shows the location of the markers they used to track the equine's jaw movements.

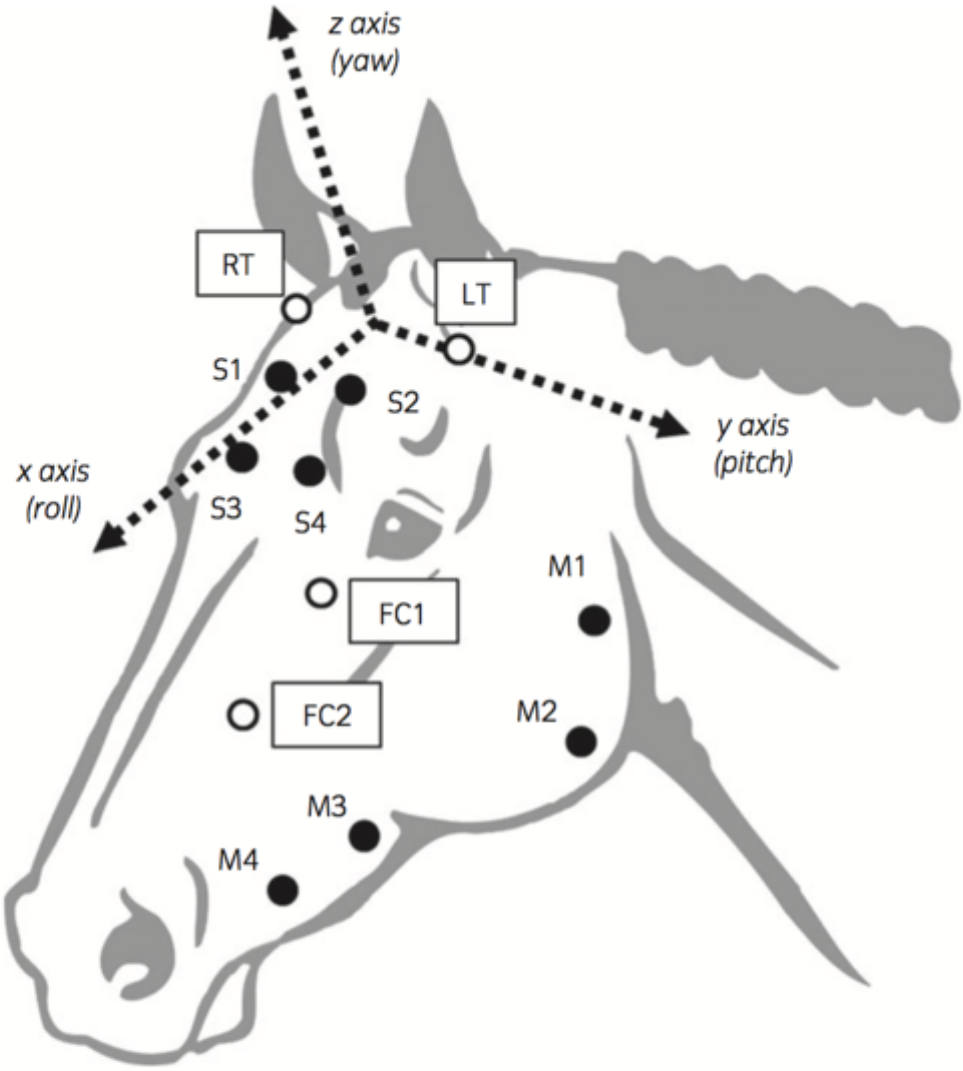

Fig 1: Marker locations and kinematics definitions. Four tracking markers were placed on the skull (S1–S4) with 2 on either side of the midline, between the eyes. The 4 mandible tracking markers (M1–M4) were located in line with the caudal ramus of the mandible. The right and left articular tubercles of the skull (RT, LT) and 2 points along the facial crest (FC1, FC2) were located using a calibrated probe and used to generate the skull coordinate system (shown). The mandible coordinate system was defined as initially coincident with the skull coordinate system.

Figure 2.2: Equine Jaw Marker Locations from (Smyth et al., 2015, p. 2)

Marker-Based Systems: Passive

Passive Marker-based approaches use markers that only reflect light. Reflective markers are used to make it easier for the camera to detect the markers since non-reflective markers are affected by lighting conditions.

In passive marker-based systems used to detect human or animal motion, markers are placed on "the locations of characteristic parts on the human [or animal] body such as joints or the joint angles" (Kapsouras & Nikolaidis, 2014, p. 1432). Often the markers cannot be placed directly at the center of the joint so "the geometric parameters of a human skeletal system and the offsets of each marker from the center of the nearest joint are also needed to be known in the computation process" (Ayusawa et al., 2014, p.274). This information is used to populate a kinematic model of the body or to provide reference points for shape or appearance models.

Weber et al. (2008) start with an initial skeleton model, then capture joints and joint angles to track motion. Weber et al. (2008) and Meyer et al. (2014) note that although this approach is effective, matching makers to a skeletal model is time consuming. Krayevoy & Sheffer (2005) overcomes this disadvantage by building a skeletal model based on information captured from markers.

Marker-based passive systems have been used in the entertainment industry for years and are nicknamed MoCap (short for Motion Capture). Currently "Passive optical motion capture is the most accurate, flexible and common type of mocap" (Vicon, 2016, para. 17). Typically, such systems use between 6 and 50 cameras placed on the walls and ceilings of a recording studio (Bregler, 2007). A body suit is worn by the actor and reflective markers are placed on key locations (typically, the joints); it takes about 50 markers to cover the entire body (Bregler,

2007). “Triangulating the tracked two-dimensional (2-D) marker locations of each camera allows for very accurate 3-D reconstruction of the markers in the recording studio” (Bregler, 2007, p. 156).

Several commercial systems for commercial capture exist. Two of the most prominent manufacturers are Vicon and Qualisys. Both of these systems consist of multiple synchronized cameras, markers, and a computer where the data gets stored (Vicon, 2016) (Qualisys, 2016).

Qualisys has a system designed to work with equine kinematics with algorithms tuned to measure “the kinematics of the joints of the distal limb [i.e. the legs] and the spine” (Qualisys-PDF, 2012, p. 1). The spine analysis is designed to be used on an equine treadmill only (Qualisys-PDF, 2012). Figure 2.3 shows a Qualisys system being used to capture a Harness racehorse’s motion.

Figure 2.3: Capturing Harness Racehorse Motion from (QualisysPicture, 2016).

Qualisys is not the only company to produce systems for equine analysis based on marker-based techniques. Motion Imaging Corporation (MIC) developed the Equine Gait Trax system which was “designed to aid in the evaluation and assessment of gait in horses” by “utilizing a specific

set up of markers" (MI-AS, 2016, para. 4-5). It also has a version for dogs called Canine Gait Trax (MI-AS, 2016).

Most of the commercial motion capture systems use multiple cameras since "a multi-camera system is required to provide some 3D information" (Rougier & Meunier, 2010, p. 505). Single camera systems have seen less use due to challenges of "depth ambiguities, high-dimensional representation of human pose, self-occlusion, unconstrained motions, observation ambiguities, motion blurs and unconstrained lighting" (Hen & Paramesran, 2009, p. 1). Even with these difficulties, there have been several studies using a single camera system. One area that has been particularly successful is head-pose tracking using a single-camera system (Tariq & Dellaert, 2004). Since "the human head is an easy human body part to track in a scene as it is usually visible from different camera points of view and its shape is relatively simple" (Rougier & Meunier, 2010, p. 505).

Research has emerged that combines smartphones with marker-based systems. Lee et al. (2015) uses a smartphone mounted on a robot to navigate a known indoor course with the use of QR markers mounted on the ceiling. The smartphone serves as the QR marker reader and communicates over WiFi to a computer which maintains the database of the next QR location based on the current QR marker position (Lee et al., 2015).

Kim et al. (2015) created a system called SmartGait which uses a smartphone's camera to assess the gait of a human. With SmartGait, users wear two green colored circular makers on the top of their shoes. The smartphone is outfitted with a specialized 90 degree, wide angle lens produced by Hilo (Hilo, 2016). The smartphone is mounted to a customized belt worn around the waist with the camera lens pointed down toward the feet. "The on-board software is designed to avoid

computational demand on the smartphone's CPU and is only used for data collection and providing a raw assessment of SL [Step Length], SW [Step Width], ST [Step Time], and speed" (Kim et al., 2015, p. 140). The data is recorded so off-board analysis can be performed to increase accuracy (Kim et al, 2015).

Cha et al. (2014) present a system that is similar to the traditional marker-based systems out there. Their system, named SmartPTA, uses two smartphones and a notebook computer that functions as the server. The user wears 13 reflective markers. The smartphones are mounted on independent tripods with their cameras aimed at the subject wearing the markers. "Each smartphone transmits detected 2D marker positions to the server using Wi-Fi at 15 frames per second" (Cha et al., 2014, p. 44). The server then reconstructs the 3D pose of the subject based on the received marker positions (Cha et al., 2014).

Hybrid Systems

Hybrid systems combine both optical and non-optical sensors. Tao et al., (2007) used a hybrid system of inertial sensors and optical cameras to track a human's arm movement for providing feedback to a patient performing home-based physical therapy. Olsen et al. (2012) used a similar approach to model equine's movements, employing inertial measurement sensors mounted on an equine and optical motion-capture cameras to detect variations in equine gaits.

INFORMATION ACQUISITION

In its most fundamental form, tracking can be defined as capturing the motion of a subject over time. Motion capture in its purest sense dates back to the 1870s. Eadweard Muybridge used multiple cameras to take photographs of a galloping horse (Back & Clayton, 2013). Étienne-

Jules Marey used a single camera to "record several phases of movement on one photographic surface" (Back & Clayton, 2013; Étienne-Jules_Marey, 2016, p. 2). With the advances of technology, tracking has become more sophisticated but the fundamental goal of capturing a subject's motion over time remains unchanged since Muybridge and Marey's time.

Segmentation

The first step in the tracking phase is concerned with separating the main subject of the image from the remainder of the image. This is especially important in markerless systems. There are several different approaches to accomplishing this: background subtraction, motion-based segmentation, appearance-based segmentation, shape-based segmentation, and depth-based segmentation (Moeslund et al., 2006). Stauffer & Grimson (1999) introduced a background subtraction approach that uses a mixture of Gaussians to model each pixel and then use on-line approximations to update the model. Motion-based segmentation approaches are based on the idea that the differences between one frame and the next is a moving object. Appearance-based segmentation approaches focus on a specific aspect of the object. Approaches in this category assume the appearance of a human is different from the appearance of the background if using a markerless system. Both active and passive marker-based systems would be considered appearance-based segmentation as the subject of the image is segmented based on the marker's appearance. Shape-based segmentation assumes the object of interest's shape is different enough from the background as to be distinguishable. Depth-based segmentation requires multiple cameras (unless using a depth sensing device such as the Kinect) and relies on the concept that the object of interest will stand out in a 3D environment (Moeslund et al., 2006).

Pose Estimation

After the subject is separated from the background the next step is pose estimation. Pose estimation is the technique of determining the subject's position and orientation in an image and "is the process in which the configuration of body parts is estimated from sensor input" (Poppe, 2007, p. 4).

Tracking adds the temporal component to pose estimation because in tracking poses are estimated from frame to frame (Poppe, 2007). This tracking of poses from frame to frame is how the motion of the subject is captured. Pose estimation and tracking can be performed for the entire body or for just a segment of the body. The algorithms for performing pose estimation can be divided into three areas: model-free, indirect use of a model, and direct use of a model (Moeslund & Granum, 2001; Moeslund et al., 2006).

Model-Free

Model-free techniques do not rely on a predefined description, but, instead use data collected from sensors to infer a model. The model-free approach consists of two different methods for determining the pose: probabilistic assembly and training.

Probabilistic assembly of parts entails detecting the individual parts of the subject and then assembling them into the pose (Moeslund et al., 2006). Ramanan and Baker (2011) uses this approach by assuming the human body can be roughly modeled as a series of rectangles. In each frame, candidate body segments are identified and grouped together to form a rough block diagram of a human.

The second technique, training a system with truth data to recognize poses of the subject, involves a database of samples that is compared with the current image. Howe et al. (1999) reconstruct a 3D motion from video taken from a single camera. Training data gathered from a 3D motion capture system (like VICON or Qualisys discussed above) provided "frame-by-frame 3D coordinates for 20 tracked body points at 20-30 frames per second" (Howe et al., 1999, p. 821). They assembled the data into snippets, each containing a vector of the "positions of each tracked body point in each frame of the snippet" (Howe et al., 1999, p. 822). They used the training data to interpret the 3D motion of the subject given a 2D image from a single camera (Howe et al., 1999).

Indirect Model Use

In indirect model use, a model is used as a look-up into a database where more information can be found. For example, "[a] simple human model is the aspect ratios between the various limbs which may be used to guide the pose estimation" (Moeslund & Granum, 2001, p. 244).

Shubert et al. (2015) use such an approach to perform automatic initialization of a skeleton tracked by a marker-based system. They use the first frame to query a large database of previously observed poses to narrow down the possible poses to approximately 100; then they select the pose that has the best tracking performance over that sequence of frames (Shubert et al., 2015). They tested their system on sheep and humans and found their system to be accurate within 10 centimeters 81.55% of the time although the initialization took approximately anywhere between 3-9 minutes to determine the best starting pose (Shubert et al., 2015).

Direct Model Use

Direct model use consists of techniques that explicitly use a subject's kinematics, shape or appearance to determine the pose of the subject. Moeslund et al. (2006) indicate that this is the most widely used research technique.

Abson & Palmer (2015) demonstrate the complexities involved in creating a direct model for use in a marker-based system by creating a whole body equine skeletal model created from a detailed biomechanical study of the equine. Suggesting that "a sound understanding of the animal is required to inform bone creation, marker placement as well as marker and bone constraint relationships," the researchers studied anatomy books and consulted with veterinarians to gain an understanding of the equine's skeleton to determine the key joints whose motion needs to be captured to accurately model the equine's movement (Abson & Palmer, 2015, p. 344). They also took into consideration the effects skeletal motion has on the movement of fat, muscle, and skin and used this data to inform them on the placement of markers for tracking joint movement so as to minimize extraneous movement caused by matter between the marker and the underlying bone (Abson & Palmer, 2015). They decided 75 marker locations were required to capture the whole body movement of the equine – although 3 had to be removed to accommodate a rider; 75 markers allowed for two to three markers per bone segment which provided redundancy in case one of the markers fell off during the data capture (Abson & Palmer, 2015). Figure 2.4 below shows where they placed markers on the horse.

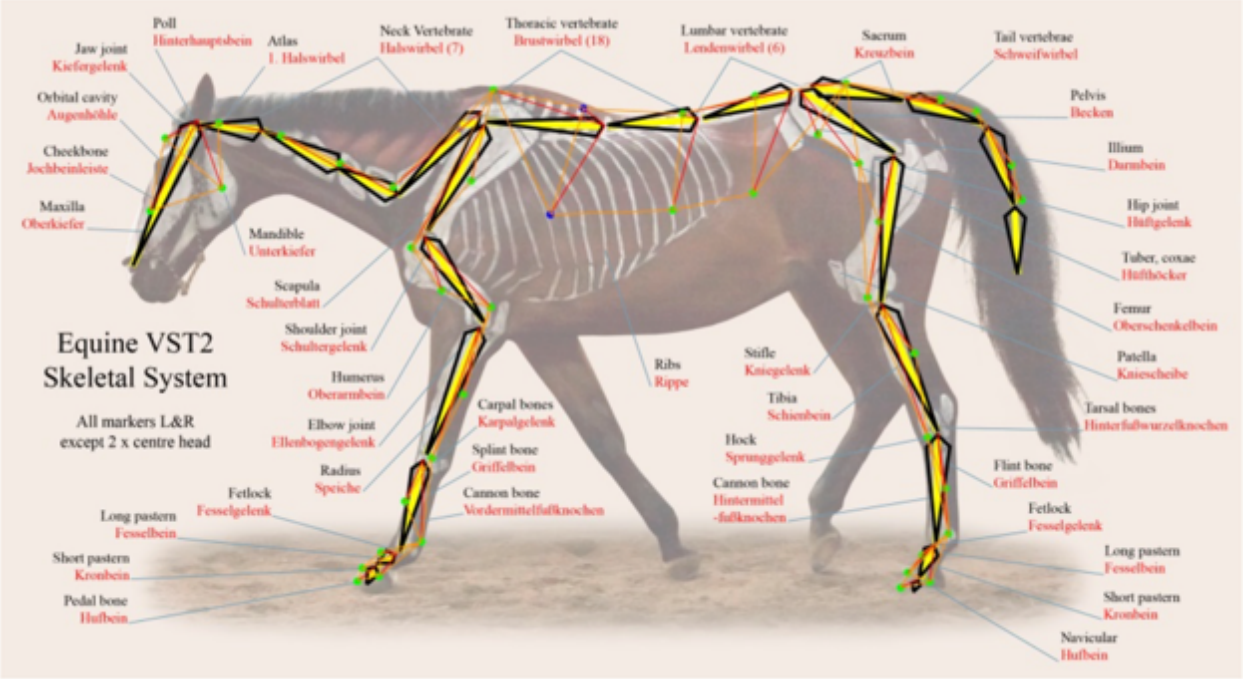

Figure 2.4: Equine Marker Locations from (Abson & Palmer, 2015, p. 347)

Smartphones

Typically, smartphone-based applications for tracking rely on internal built-in sensors such as the GPS and accelerometer to track a subject's movement; however, there has been some research done using the smartphone's camera to detect changes in the environment (including motion).

Roters et al. (2011) use video taken by a smartphone's camera to help a vision-impaired pedestrian to know when it is safe to cross the road based on the color of the traffic light. They discuss three aspects of smartphones that limit the application of traditional computer vision algorithms: "The resolution of the capture device is relatively low. Mobile devices often provide only poor image quality, e.g., falsified colors and unsharpened images due to automatic white

balance and auto focus. Computation power and memory resource are restricted. (Roters et al., 2011, p. 1498)"

Shang et al. (2012) proposes a system which uses smartphones networked together to detect the location of a subject. They propose an algorithm to estimate the location of a subject based on a single image taken from a calibrated smartphone's camera. When the image is taken, the GPS reading of the phone and the phone's orientation (based on the internal compass) is recorded. After the image is taken, the user draws a bounding box around the subject in the image. The algorithm computes the location of the subject based on the image, GPS reading, orientation, bounding box, and true size of the subject (Shang et al., 2012).

Wang et al. (2012) agree with Roters et al. (2011) assessment that "due to the limited computation power of smartphones, complex algorithms requiring high time complexity and space complexity will not fit" (Wang, 2012, p. 693). They developed a lightweight computationally efficient subject tracking algorithm that runs on a smartphone. Their system targets "vehicles or rigid moving objects that are at about the same level as the smartphone camera" (Wang et al., 2012, p. 697). Similar to the approach described by Shang et al. (2012), the user draws a bounding box in the first frame around the subject to be tracked and the algorithm assumes the physical size of the subject is known. The algorithm draws a bounding box around the tracked subject in subsequent frames and employs this information to estimate a trajectory and velocity of the tracked subject (Wang et al., 2012).

RECOGNITION

Recognizing Behavior

The goal of recognition is to determine what the subject is actually doing. Recognition techniques can be broadly divided into static recognition and dynamic recognition. Static recognition techniques primarily use spatial data while dynamic recognition techniques primarily rely on temporal characteristics (Moeslund & Granum, 2001). The distinction between the two techniques is not pure; algorithms consider both spatial and temporal characteristics.

More interesting is how the recognition is done. Some of the more common approaches are action hierarchies, scene interpretation, holistic recognition approaches and recognition based on body parts (Moeslund et al., 2006).

An action hierarchy is the concept that each high-level activity can be broken down into smaller actions, which are, in turn, broken down into action primitives. For example, the activity of playing tennis has as one if its actions return the ball whose action primitives could be forehand, backhand, run left, run right, etc (Moeslund et al., 2006).

Scene interpretation are techniques that "consider the camera view as a whole and attempt to learn and recognize activities simply by observing the motion of objects without necessarily knowing their identity" (Moeslund et al., 2006, p. 111).

Holistic recognition approaches consider the entire subject and attempt to recognize what the subject is doing based on the entire body. In contrast, recognition based on body parts

techniques focus on recognizing individual parts and then determining what the subject is doing based on what the individual parts are doing (Moeslund et al., 2006).

Miller (2012) suggests that "smartphones offer huge potential to gather precise, objective, sustained, and ecologically valid data on the real-world behaviors and experiences of millions of people where they already are, without requiring them to come into labs" (Miller, 2012, p. 221). The author argues that not only could context-aware surveys be presented to the users based on their GPS location but behavioral information could be gathered from the variety of sensors built into the smartphone. As examples, he provides a table of 13 different research projects from 2005 – 2011 that uses the smartphone to gather behavioral data of the human user such as emotional recognition from microphone input, GPS correlation and ambient noise monitoring correlated with self-entered mood report and combining GPS and GIS data to predict the person's environmental impact (Miller, 2012).

There have also been studies using videos taken by smartphones to analyze animal's movement from which the researchers can then manually recognize the animal's behavior. Pittman & Ichikawa (2013) used two commercial applications, SwingReader (bought out in 2014 and now sold under the name of Hudl and Sports Motion Analyzer to track the movement of zebrafish to quantify behavioral changes following psychoactive drug exposure (Crunchbase, 2016; Sports Motion Analyzer, 2016). Their work demonstrates the feasibility of using a video from a smartphone and an application running on the same smartphone to quantify the behavior of an animal (Pittman & Ichikawa, 2013).

Conklin et al. (2015) use OpenCV and Python to develop a video tracking system to track animal movement. Their system processes video from smartphones, laptops, or webcams but runs on a computer. It detects motion by first converting the frames to grayscale and then performing a pixel-by-pixel subtraction. This operation results in an image with black pixels representing no movement detected and white pixels representing movement. The resulting data can be analyzed to determine the behavior of the animal. They applied their system to determine what type of food do birds like by counting the number of times movement is detected at four different food offerings. To do this, they pointed a webcam at 4 different bowls: one contained raisins, the next contained sunflower seeds, the third miscellaneous small seeds, and the fourth peanuts. Detection of movement as well as timing of movement was recorded for each bowl. Based on the number of seconds of movement detected for each dish, it was determined the birds prefer the miscellaneous small seeds. They also used their system to quantify the movement of adult zebrafish treated with ethanol, a Siberian dwarf hamster navigating a maze, and how larval zebrafish respond to tricaine and heat. This approach is sensitive to any source of "motion artifacts, such as moving shadows, glare, changes in light intensity, or warping in the image resulting from lens curvature" (Conklin et al., 2015, A120-A121).

Anomaly Detection

Not only is it necessary to detect behavior, but it is imperative that normal behavior be distinguished from abnormal behavior. "Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior" (Chandola et al., 2009, p. 15:2). Numerous anomaly detection methods have been developed; some anomaly detection techniques are tailored to a particular domain while others are generic and can be used across multiple domains (Chandola et al., 2009; Gogoi et al., 2011). Although generic techniques do exist, there

is not one anomaly detection method that works for all situations (Hodge & Austin, 2004). Researchers must carefully consider the domain of their research area and weigh the benefits and drawbacks of each anomaly detection method before selecting the best one.

Shown in Figure 2.5 are the three main components (Application Domain, Research Area, and Characteristics to Consider) that go into selecting which anomaly detection technique to use. The Characteristics to Consider area is further divided into the key characteristics a researcher must contemplate when selecting an anomaly detection technique.

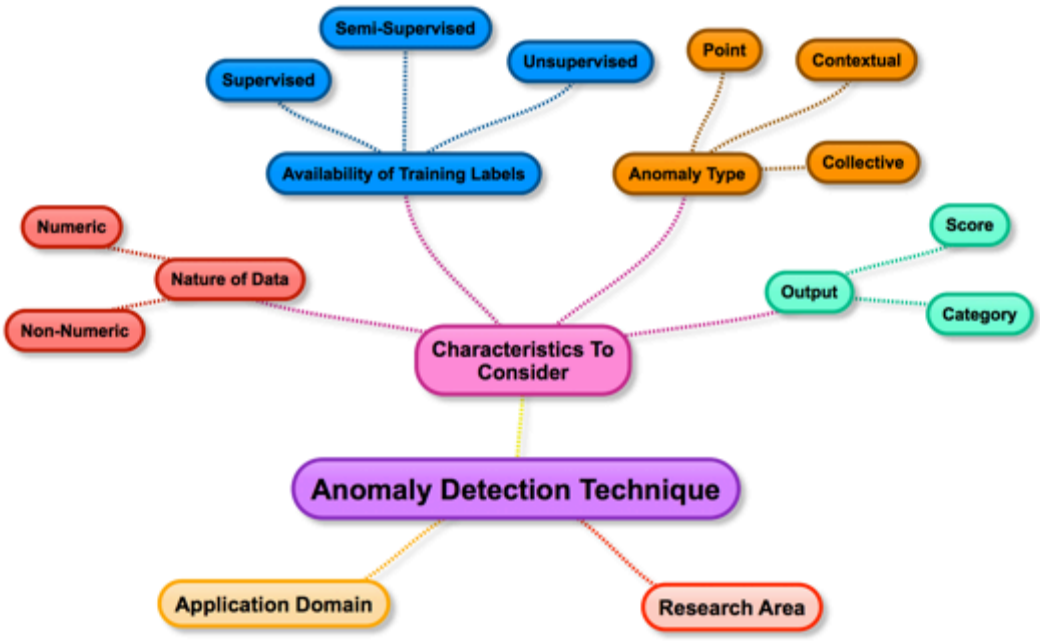

Figure 2.5: Anomaly Detection Techniques (adapted from (Chandola et al., 2009, p. 15:4)

Application Domain refers to the type of system anomaly detection technique is employed. Some examples of common systems utilizing anomaly detection techniques include fraud detection, network intrusion detection, health monitoring, etc (Hodge & Austin, 2004; Song et al., 2007).

Research Area is the particular focus area of the researcher selecting the anomaly detection technique. Some examples of research areas commonly using anomaly detection techniques include machine learning, statistics, neural networks, and data mining (Chandola et al., 2009; Hodge & Austin, 2004).

The Characteristics to Consider area represents the key characteristics to consider when selecting an anomaly detection technique and are divided into four regions: Output, Availability of Training Labels, Anomaly Type, and Input (Chandola et al., 2009).

The output of an anomaly detection technique can vary based on whether the technique defines a hard or soft boundary (Hodge & Austin, 2004). In a hard boundary system, the anomaly techniques categorize each output instance as either normal or abnormal. This is helpful when all the system cares about is whether something falls above or below a predetermined threshold. Other techniques use a soft boundary where the degree of abnormality is recorded; these techniques determine the likelihood each output is an anomaly and produce a ranked list of anomalies (Gogoi et al., 2011). A ranked list would be important in a system where it's hard to determine if an output is truly important or not and relies on the user to investigate. With a ranked list, the user knows what items he or she should focus on first. The ranked list concept can be combined with a threshold – for example the system would only show the user the top 10 anomalies detected by the system (Gogoi et al., 2011).

Anomalies can be separated into two broad types: simple anomalies (also called point anomalies) and complex anomalies that are further sub-divided into contextual anomalies and collective anomalies (Chandola et al., 2009; Gogoi et al., 2011). Point anomalies are single items detected

as outliers. In contextual anomalies (also called conditional anomalies), the context of the situation is taken into account to determine whether an item is an anomaly or not. For example, a rapid heart rate would not be considered anomalous for a person running a sprint but would be considered anomalous for a person sleeping. Collective anomalies occur when the combination of points is considered anomalous but the individual points forming that combination might not be considered anomalous when viewed independently (Chandola et al., 2009) (Gogoi et al., 2011).

Availability of Training Labels is categorized into three areas – supervised, semi-supervised, and unsupervised (Chandola et al., 2009; Gogoi et al., 2011; Hodge & Austin, 2004). The differences between these categories are the amount of training data required for the algorithms to work. With supervised techniques, training data is available for both normal and abnormal behaviors. Semi-supervised techniques assume training data is available for either the normal behaviors or the abnormal behaviors but not both (Chandola et al., 2009; Hodge & Austin, 2004). Typically, normal behaviors are used for the training data in the semi-supervised techniques; using only the abnormal behavior for the training data is rare (Hodge & Austin, 2004). Unsupervised techniques do not assume training data is available but do have the implied assumption that normal data is more frequently observed than abnormal behavior (Chandola et al., 2009; Gogoi et al., 2011).

Finally, researchers must consider the nature of the input data. Input data can be numeric or non-numeric (also called symbolic) (Agyemang et al., 2006). Numeric data can be either single points or ranges. Non-numeric is typically textual or imagery data. The researcher needs to consider what input data is available before selecting an anomaly detection technique. For

example, if the researcher implements a numerical technique but then has only image data available, the researcher must then implement the added step of converting the image data to a numerical representation or switch to a different technique.

Detecting Abnormal Behavior

Although anomaly detection techniques are typically thought of as only applying to computer-based applications such as fraud detection and network intrusion detection, the basic concepts apply for algorithms detecting abnormal behavior in both humans and animals. Zanero (2004) considers "anomaly-based intrusion detection in the more general frame of behavior detection problems" (Zanero, 2004, p. 658). He proposes a general framework for performing behavioral detection using anomaly detection methods. The three high-level steps of his framework are: "1) Specify which kind of displays of behavior we can detect and build appropriate sensors for detecting them. 2) Choose an appropriate model for representing the behavior. 3) Set thresholds and logics that help us extract useful information from the observed behavior" (Zanero, 2004, p. 660-661).

Pannell & Ashman (2010) present a similar approach based on user models. They build a model based on each individual's behavior while using a computer and focus on "some user feature [i.e. behavior], such as typing habits and Web page usage, more so than application-specific features which only indirectly reflect user activity" (Pannell & Ashman, 2010, p. 208). To detect anomalous behavior, they compare the user's current activity to the activity reflected in the user model. Although they apply this approach to an intrusion detection system, they argue it could also be beneficial in health monitoring activities such as "detecting degradation in a user's skills over time" (Pannell & Ashman, 2010, p. 208).

Roshtkhari & Levine (2013) present a model-free, unsupervised learning algorithm to detect anomalous behavior in videos. They use "densely sampled spatio-temporal video volumes (STVs)" to "construct a set of behavior patterns for each pixel" (Robert & Levine, 2013, p. 2611). Then they use this information to classify normal (aka dominant behaviors) in a video as those occurring frequently and separate them from anomalous behavior which they define as behavior occurring infrequently. "A limitation of the current approach is that it does not account for trajectories and hence, long term behaviors are not learnt" (Roshtkhari & Levine, 2013, p. 2618).

Nie et al. (2009) developed an algorithm to detect in real-time the abnormal behavior of scratching in laboratory mice. Their algorithm is designed to distinguish scratching from other behaviors displayed by mice such as grooming and rearing. The algorithm is intended to be used in a very controlled environment which includes a transparent acrylic cage, a high-speed vision system operating at 240 fps, and an IR illuminator which the cage sits on.

CHAPTER THREE: RESEARCH APPROACH

OVERARCHING GOAL

The overarching goal of this research was to find an easily-accessible, low-cost way to automate the detection of abnormal behavior. Smartphones are ubiquitous in today's society, but are associated with humans and human behavior – not animals. We wanted to use an unmodified commercial-off-the-shelf (COTS) smartphone to detect when an animal – an equine in this case – exhibits abnormal behavior. This required novel approaches as the behavior between a human and an equine are different with each requiring different models to represent their normal and abnormal behavior. This research focused on modeling an equine in such a way as to detect its behavior using only a smartphone.

OBJECTIVES

The primary objectives in support of the research's overarching goal were to model equine behavior in such a fashion as to:

1. Model behavior of the animal (Develop the Model)
2. Use ubiquitous sensors to acquire information on the animal (Acquire Information)
3. Analyze data and identify in near real-time behavior that deviated from normal (Identify Behavior)

In our research the first objective was performed in the modeling phase. The second and third objective were performed in the data acquisition/behavior recognition phase. These three objectives can be broadly divided into two categories as follows:

1) Modeling
    - First Objective: Represent the Model
2) Simulation
    - Second Objective: Acquire Information
    - Third Objective: Identify Behavior

Schwendimann (2010) describes a model as "a product (physical or digital) that represents a system of interest (Schwendimann, 2010, para. 1). A model is similar to but simpler than the system it represents, while approximating most of the same salient features of the real system as close as possible." He describes a simulation as the "process of using a model to study the behavior and performance of an actual or theoretical system" (Schwendimann, 2010, para. 3). In its simplest representation a simulation can be viewed as the following:

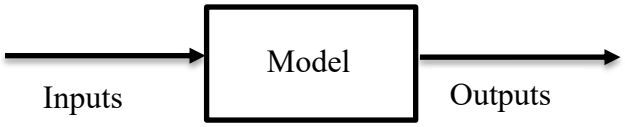

Figure 3.1: Diagram of a Simulation

Overlaying our objectives on Figure 3.1 gives us the following:

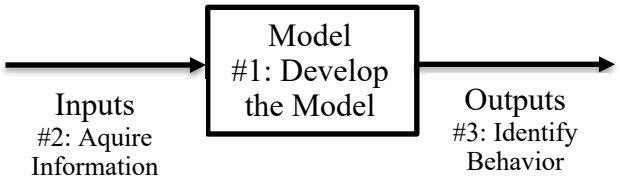

Figure 3.2: Diagram of a Simulation with Overlaid Objectives

MODELING

First Objective: Develop the Model

Modeling Questions

The questions that informed our model building process were as follows:

- What is the platform the model will run on?
- What is the physiology and behavior of the subject to be modeled?
- What is the purpose of the model?
- What is the environment of the subject?
- What data is available to inform the model?

The answers to these questions greatly influence the model that can be built. None of these questions can be considered independently because a decision made in one area can impact answers in the other areas.

Answering the question of what type of platform the model runs on constrains the development of the model. A model that runs on a high performance computing platform has much more processing power available to it than a model running on a smartphone. The high performance computing platform would be able to run more processing intensive algorithms than the smartphone.

To answer the second question a study of the subject must be done. Depending on what the purpose of the model is this study might be extremely detailed or more general. For example, to build a model of a cat's tail a researcher could observe a cat moving its tail and then mimic the

observed behavior in a model as a simple string. Alternatively the cat's tail could be modeled at the cellular level depicting which nerves fire on which muscles to cause the tail's movement. Modeling at the cellular level requires a more detailed study into the subject's kinematics than a simple string approach. Which approach chosen depends on how the model is going to be used (i.e. the model's purpose).

The environment of the subject guides the model to be built. To build an aerodynamics model of an aircraft, the environment the aircraft is going to operate in greatly influences the model. If the aircraft is going to be used in outer space, a different model will be needed than if the aircraft is going to be used in the Earth's atmosphere.

Similarly, the data available to inform the model must be taken into account. For example, if the only sensor available to gather real-time data on the subject is a smartphone building a model that requires input data at the sub-cellular level is not realistic.

Model Representation

Once the subject to be modeled, its associated behaviors, and physiological characteristics are known, they have to be combined into a cohesive usable whole. One methodology for doing this is to use a framework. PCMag (2018) describes a framework as "a set of common software routines that provides a foundation structure for developing an application. Frameworks take the tedium out of writing all the program code for an application from scratch (PCMag, 2018, para. 2)." For this research we took the novel tactic of using a gaming framework for our modeling approach. As the name implies, a gaming framework is typically used for developing games and is not traditionally associated with behavioral modeling and certainly not behavioral modeling of

an animal. However, upon closer inspection a gaming framework provides the key components needed for behavioral modeling. One of the key components provided by most gaming frameworks is a Component/Entity System (C/ES).

According to the Entity Systems Wiki (2014), a C/ES is based on the principle of composition. An entity is a container into which components can be added. These components represent the behaviors, looks and/or data associated with the entity (Entity Systems Wiki, 2014). House (2012) uses the analogy of a key to explain the relationship between entities and components. He explains: "the teeth of our entity key is the components that make it up. You can tell entities apart by their ID, even if they have the same teeth (House, 2012, para. 3)."

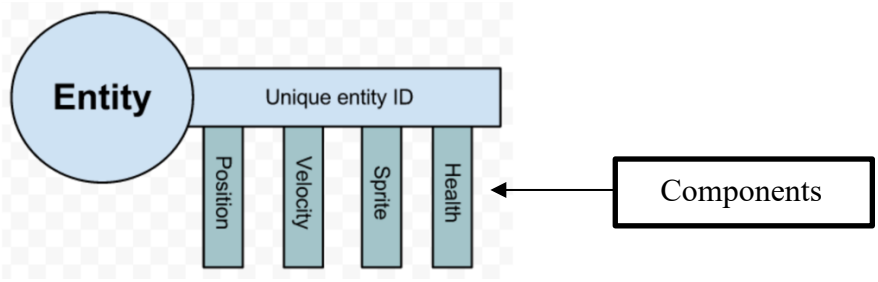

Figure 3.3: Component/Entity Relationship (House, 2012, Fig. 1)

Since our specific platform was an iPhone 6, we used Apple's gaming framework, GameplayKit. Apple's GameplayKit is "a general architecture for designing composable, reusable gameplay logic" (Apple, 2018b, para. 5). As part of GameplayKit, Apple provides a C/ES consisting of a GKComponent and a GKEntity. Although developed for use with games, Apple's generic architecture for designing composable entities was perfect for behavioral modeling of animals. The animal was modeled as a GKEntity. The corresponding behaviors to be detected and the actions to take based on those behaviors were modeled in separate GKComponents.

To model behavior with a C/ES a researcher has to:

1) Determine what actions each component will take when it is executed
2) Come up with a theoretical behavioral pattern that is representative of each behavior to be modeled

The actions each component performs when it is executed is based on the purpose of the model. Examples of such actions are logging data to a file, playing a sound, or sending a text message.

Developing the theoretical behavioral pattern requires determining what each of the modeled physiological characteristics of the subject is doing for each behavior to be modeled. This data is gathered via observation and analysis when preexisting data is not available. How this observation and analysis is performed depends on the purpose of the model. For example, if the purpose of the model is to distinguish how a dog wags its tail based on certain stimuli, a study such as that described in (Quaranta et al., 2007) could be performed. Quaranta et al., (2007) placed thirty dogs one at a time in a box and exposed them to different stimuli. A video was taken of how the dog's tail reacted for each stimulus. The researchers analyzed the video data and determined a composite amplitude of tail wagging that corresponded to each stimulus. This amplitude data forms the theoretical behavioral pattern.

SIMULATION

Second Objective: Acquire Information

Objective 2 is associated with acquiring information from sensors and inputting this information into a model. Google's dictionary describes a sensor as "a device that detects or measures a

physical property and records, indicates, or otherwise responds to it" (Google, 2018, para. 1). Raw data from sensors is either fed directly into a model or sent into a processing routine to reduce the raw data into a format useable by the model. Our proof of feasibility was focused on visual acquisition, so the input for our model consisted of images. Typically an image's raw data contains both areas of interest and non-interest and must be processed before it is useable by the model. For example, if the purpose was to build a model to count how many times a basketball bounced, the basketball (area of interest) must first be separated from the background scenery and from the player's hands bouncing the ball. Then the basketball's information could be fed into a model where the bouncing behavior was counted.

One of the critical pieces of using image data for modeling is to determine at what interval (i.e. framerate) with which to sample the data. Per Pueo (2016) to calculate the minimum framerate required for perfect detection accuracy one needs to know the velocity of the object and the size of the object. The smaller the object and the faster the velocity the higher the minimum framerate required. Pueo's (2016) formula for calculating the minimum frame rate is shown below:

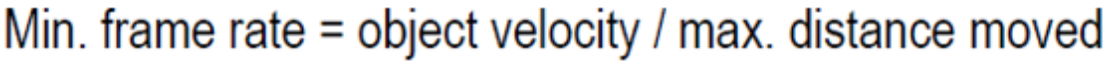

Figure 3.4: Minimum framerate formula (Pueo, 2016, p.57)

A corollary question is what is the slowest framerate the model can receive input and still distinguish normal from abnormal behaviors. This question is more experimental in nature as there is not a formula available to calculate it. The slowest framerate is dependent on the

domain of the research and on the subject's specific behaviors. The slowest framerate will fall somewhere between 0 and Pueo's (2016) minimum framerate. Since the purpose is to calculate the slowest possible framerate, we start at 0 and work our way up and evaluate the accuracy of the output at each increase in framerate. As the framerate is increased towards Pueo's (2016) minimum, the accuracy should increase. Once the accuracy has reached a pre-determined threshold the slowest possible framerate has been achieved. The pre-determined threshold is set based on the purpose of the system. For example, a proof-of-feasibility system will require a much lower accuracy threshold than a production system.

Another key component provided by most gaming frameworks is a methodology to receive inputs in real-time, update the state of the system based on those inputs, and then take actions based upon the inputs received. This process is what Trochim (1989) describes as the "acquisition of an observed pattern" (Trochim, 1989, p. 1). While the simulation is running, it is receiving processed inputs from the sensors at pre-determined interval and feeds this data into the gaming framework. The gaming framework this updates the state of the simulation based on these inputs. To determine what actions to take based on the updated state, Apple's gaming framework provides a built-in rule-based system. According to Lengyel (2015), "a rule-based system is a series of if-then statements that utilizes a set of assertions, to which rules are created on how to act upon those assertions" (Lengyel , 2015, p. 1). To develop a rule, a series of assertions (also called predicates) are developed and then combined to form a rule. A simple example of how this might be used in a game is as follows:

If ((player's number of lives > 1) AND (players health is < 10%)) then
decrement the player's entity number of lives.

In the above example (player's number of lives > 1) and (players health is < 10%) are examples of assertions and the entire statement is the rule to be evaluated.

Third Objective: Identify Behavior

According to Trochim (1989), pattern matching is the "specification of a theoretical pattern, the acquisition of an observed pattern, and an attempt to match the two" (Trochim, 1989, p. 1). The theoretical behavioral pattern was developed during objective 1, the acquisition of an observed behavioral pattern was accomplished in objective 2, objective 3 deals with the final part of Trochim's definition - "an attempt to match the two" (Trochim, 1989, p. 1). Similar to objective 2, a rule system can be used to accomplish this objective. The general structure for such rules is as follows:

If the theoretical pattern equals the observed pattern then

A pattern match has occurred

Once the rule has been successfully executed (i.e. a match has been found), the entity's component corresponding with the matched behavior is called. The gaming framework transfers control to the component to execute its actions defined during the modeling phase.

RESEARCH CONTRIBUTIONS

The two primary contributions of this research were:

1) Employed a gaming framework on a relatively low-power/low-capability devices to monitor animal behavior.
2) A means to describe and detect anomalous animal movement using a rule-based pattern matching methodology.

CHAPTER FOUR: RESEARCH VALIDATION

RESEARCH HYPOTHESES

This research has two competing hypotheses. The null hypothesis (H0) assumes the smartphone cannot detect nor distinguish between normal or abnormal behavior while the positive hypothesis (H1) assumes the smartphone can detect and distinguish between normal and abnormal behavior. Both hypotheses can be stated as follows:

> H0 – Smartphones cannot detect an equine pawing in a stall and cannot distinguish the pawing from an equine walking around or standing still in a stall.

> H1 – Smartphones can detect an equine pawing in a stall and can distinguish the pawing from an equine walking around or standing still in a stall.

LIGHTING

Lighting conditions played a significant role in the computer vision techniques utilized for this application. Shifting shadows and changes in the sun's illumination can throw the algorithms off. For example, Figure 4.1 shows a picture taken in the stall at 8am on a bright sunny morning. Figure 4.2 shows a picture taken in the same stall with the same camera at 8pm that evening when the sun is gone over the horizon and the barn's electric lighting is providing most of the illumination. Because of these changes, the software had to be manually calibrated each time the program was run.

Figure 4.1: Lighting in Stall at 8am

Figure 4.2: Lighting in Stall at 8pm

MODELING PHASE

Equipment Used for Modeling

The modeling effort used the following equipment:

- One iPhone 6
- Flexible Tripod Mount
- Lightning to USB transfer cable
- MacBook Pro laptop

The iPhone 6 was developed by Apple and released on 19 September 2014 (iPhone, 2016). It features an 8-megapixel back facing camera (Apple, 2016). We used an iKross Universal smartphone Flexible Tripod Stand Holder to mount the iPhone 6 to the stall (Amazon.com, 2016). The iKross has bendable legs that can be manipulated to fit around objects (such as the top wood beam of the stall) and does not require a permanent mounting method. This flexibility allowed us to experiment with the best placement of the smartphone to capture images of the equine. The Lightning to USB transfer cable was used to transfer the video taken with the iPhone 6 to the MacBook Pro for analysis. We used a circa 2013 MacBook Pro with a 2.4 GHz Intel Core i5 processor and 8 GB of 1600 MHz DDR3 memory. The MacBook Pro was used to analyze the video and develop the model.

Steps in the Modeling Phase

The following steps were used to collect data for building the model:

1) Mount the iPhone in the stall using the iKross tripod.
2) Put equine in the stall.
3) Set the iPhone's back camera to video mode and start recording.

4) After the video has run long enough to capture the desired behavior (the equine has pawed, walked, or stood still), stop the video.
5) Download the video onto the laptop for analysis.

We analyzed the video frame-by-frame and through this analysis discovered an equine's front legs displayed a distinct pattern of movement for the pawing, walking, and standing behaviors. An equine pawing in a stall shows a pattern of no movement on the non-pawing leg with movement on the front leg that is doing the pawing. An equine walking around in the stall shows a pattern of movement on both front legs. An equine standing still in a stall shows a pattern of no movement on both front legs. These behavioral patterns were synthesized into markers. We used a 1 to represent movement on a leg while a 0 represented no movement. These markers were then combined to form a theoretical pattern for each of the behaviors:

10 = Right leg moved while left leg stayed still (Behavior = Pawing Right Leg)

01 = Left leg moved while right leg stayed still (Behavior = Pawing Left Leg)

11 = Both legs moved (Behavior = Walking)

00 = Both legs did not move (Behavior = Standing Still)

We used Apple's Component/Entity System (C/ES) to encode our model of an equine. The equine was modeled as a GKEntity. The pawing, walking, and standing still's behaviors were embedded into separate GKComponents. As discussed in chapter 3, another important part of developing a component is to determine what actions will be performed when the component is executed. For this research, each GKComponent performs a logging action and records its data to a text file.

DATA ACQUISITION/BEHAVIOR RECOGNITION PHASE

Equipment Used for Data Acquisition

The data acquisition effort used the following equipment:

- Two iPhone 6
- Custom-Built Mounting Board
- Self-Adhering bandages

To run the custom-built software and collect the validation video, two iPhones 6 were used. One iPhone 6 ran the custom-built software and the other iPhone 6 took a video of what the equine was doing while the software was running on the other phone. Two iPhones were necessary since a single iPhone cannot take video and photos with the same camera at the same time. The researcher used a handheld custom-built mounting board so the two iPhones remained lined up and stable while the software and video camera were collecting data.

This research employed 3M's VetRap ®, a brand commonly used to bandage equines because it sticks to itself without sticking to the equine's fur. A red bandage marked the right front leg and a green bandage marked the left front leg. See Figure 4.3 for an example of how the VetRap ®, was applied to the equine's legs.

Figure 4.3: Equine's Front Legs with VetRap ®

Steps in the Data Acquisition/Behavior Recognition Phase

The following steps were used to acquire the data:

6) Mount both iPhone 6s to the custom-built mounting board.
7) Wrap the equine's front legs – red for the right and green for the left.
8) Put equine in the stall.
9) Start the custom-built software and adjust for lighting conditions.
10) Set the second iPhone's camera to video mode.
11) Start the custom-built software on the first iPhone and record video on the second iPhone.
12) After the software has run long enough to detect the desired behavior (the equine has pawed, walked, or stood still), stop both the custom software and the video.
13) Download both the data and video onto the laptop for analysis.

After the data had been collected a transfer cable was used to transfer the data from the first iPhone and the video from the second iPhone to a laptop for analysis.

The software program produces three key outputs:

1) The photos taken with the iPhone camera through the custom software.
2) The output photos created by the custom software that displays the location of each leg with a red or green colored circle.
3) A text file that contains the movement pattern detected, what the algorithm determined the equine to be doing in the last second, the corresponding filename of the last photo the movement is associated with, and a summary of the movement detected to date.

An example of each of the three key outputs produced by the software is below:

1) The software took two photos, shown in Figure 4.4.

Figure 4.4: Input Photos Taken by Software Program

2) The software isolated the markers, as shown in Figure 4.5.

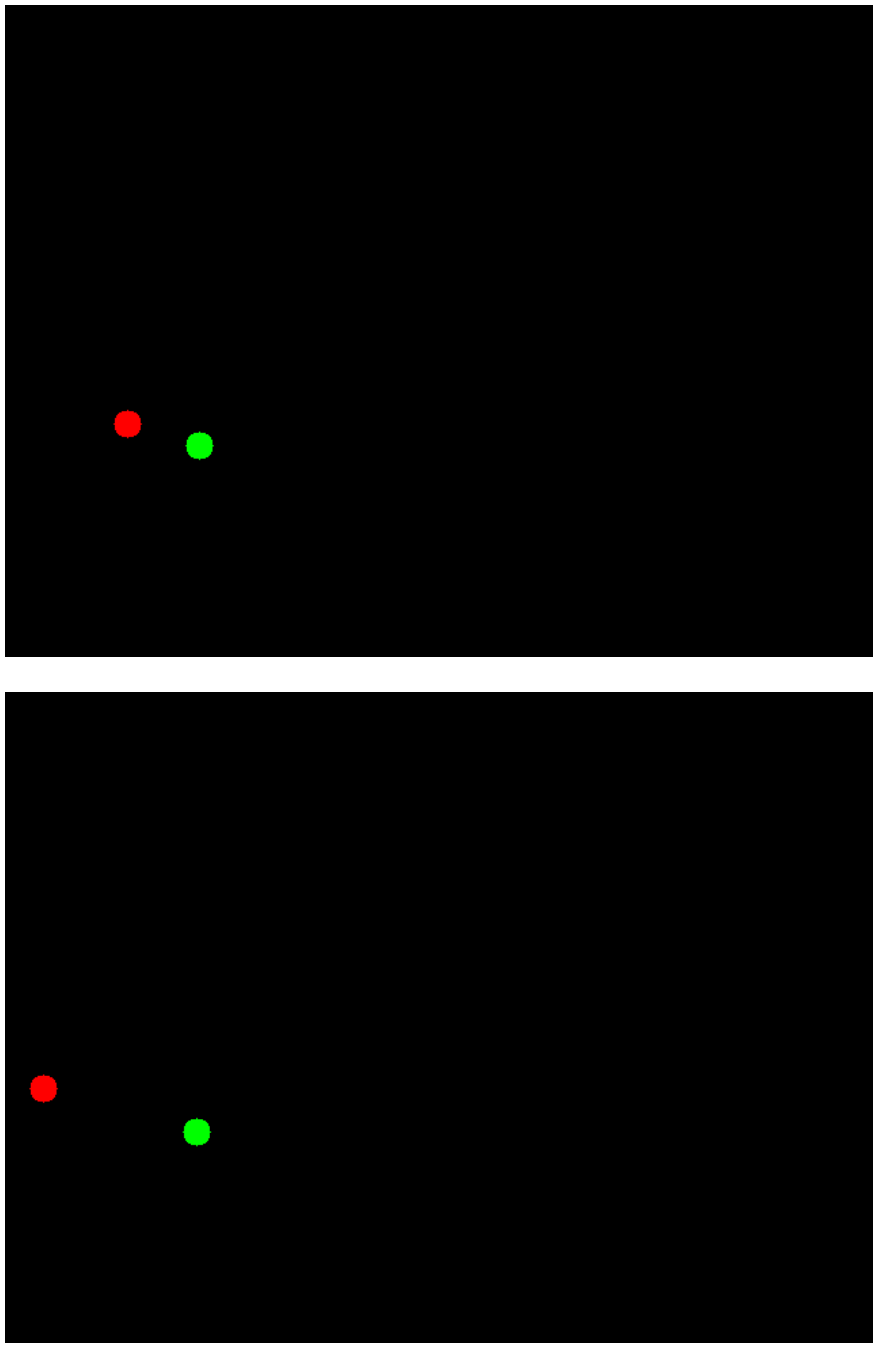

Figure 4.5: Output Photos Produced by Software Program

3) The information was codified in a text file as follows:

```
********************
Movement Pattern: 1010
In the last second: the horse pawed with the right leg

Horse Summary Stats
24.jpg
0 seconds horse has been pawing with left leg.
2 seconds horse has been pawing with right leg.
3 seconds horse has been still.
6 seconds horse has been moving.
********************
```

The video was taken using the iPhone 6 default of 1080p at 30 frames per second (fps) to record video and was stored in the Moving Picture Experts Group version 4 (MPEG-4) format the default video encoding format used by the iPhone 6.

The video provided the truth data for assessing the acuity of the software. For each one-second segment of the video, an equine expert recorded one of the following observed behaviors as shown in Table 4.1:

0	Lost Data on Both Legs
1	Lost Data on Right Leg
2	Lost Data on Left Leg
3	Horse Stood Still
4	Horse Pawed Right Leg
5	Horse Pawed Left Leg
6	Horse Moved

Table 4.1: Behavior Categories

The behaviors were corroborated separately by an equine veterinarian[2].

[2] Dr. Jennifer S. Taintor, DVM, MS, DACVIM, DACVSMR of Auburn Veterinary School

The behavior detected by the software was matched against the behavior observed in the video at each time interval, resulting in a spreadsheet along the lines of Table 4.2.

Time Interval	Observed Behavior	Detected Behavior
0:00	Horse Moved	Lost Data on Right Leg
0:01	Horse Moved	Horse Moved
0:02	Horse Moved	Horse Moved
0:03	Horse Moved	Horse Moved
0:04	Horse Pawed Right Leg	Horse Pawed Right Leg

Table 4.2: Observed vs. Detected Behavior

SYNOPSIS OF DATA – FAST & SLOW MOVING EQUINES

One Second Synopsis – Fast Moving Equine

Configuring the tracking and detection algorithm to use one second of data for the equine that moves rapidly in the stall produced the results visualized in Figure 4.6.

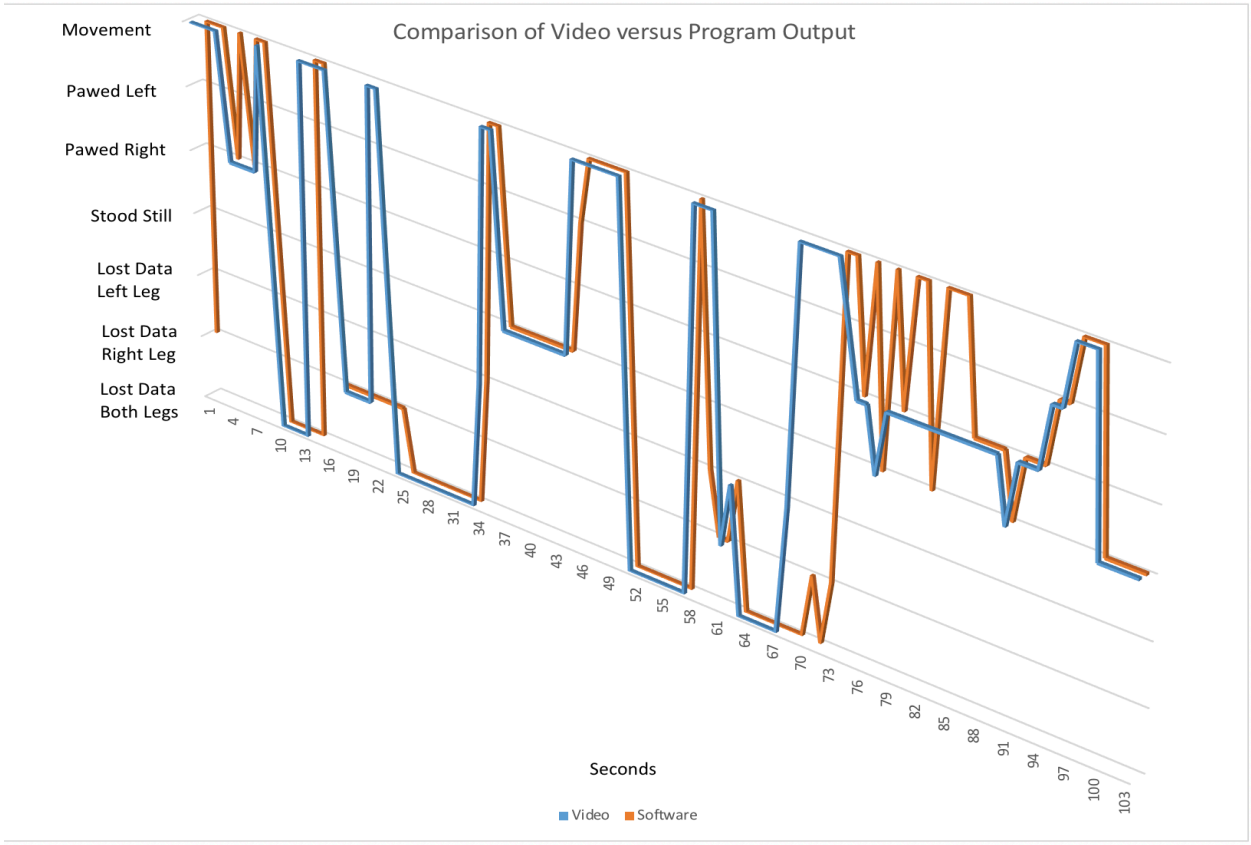

Figure 4.6: Results of One Second Synopsis – Fast Moving Equine

Twenty-five discrepancies were detected across the 103 total possible points for comparing observed and detected behavior, yielding a 24.3% error rate. These mismatches fall into broad categories as show in table 4.3. Analysis pointed to the following discrepancies between the observed and detected behaviors:

1. The detection mechanism lost track of the test subject's orientation in 13 of the 25 observation intervals.
2. The observed behavior was not accurately determined in 12 of the 25 cases. Ten of the faults diagnosed the test subject as moving when it was observed to be pawing. In the remaining two faults, the test subject was observed to be pawing and was identified as doing something else.

The analysis also indicated that the detected behavior lags the observed behavior by a slight amount, typically less than a second. The delay was caused by the time required to process the image of the test subject and locate the leg markers. The latency was minimal and did not have an impact on the functionality of the software.

Type of Mismatch (video/program)	Number of Occurrences
Movement/Lost Data (all types)	11
Pawing/Movement	10
Data Loss Type Mismatch	2
Pawing/Standing Still	1
Movement/Pawing	1
Standing Still/Movement	0

Table 4.3: Mismatch Categories for One Second Synopsis – Fast Moving Equine[3]

One Second Synopsis – Slow Moving Equine

Configuring the tracking and detection algorithm to use one second of data for the equine that moves that moves very little in the stall produced the results visualized in Figure 4.7.

[3] There are three additional potential categories for mismatches: Lost Data (all types)/Movement, Standing Still/Pawing, Movement/Standing Still. However, no mismatches in these categories occurred during my research so were intentionally omitted from this and subsequent tables.

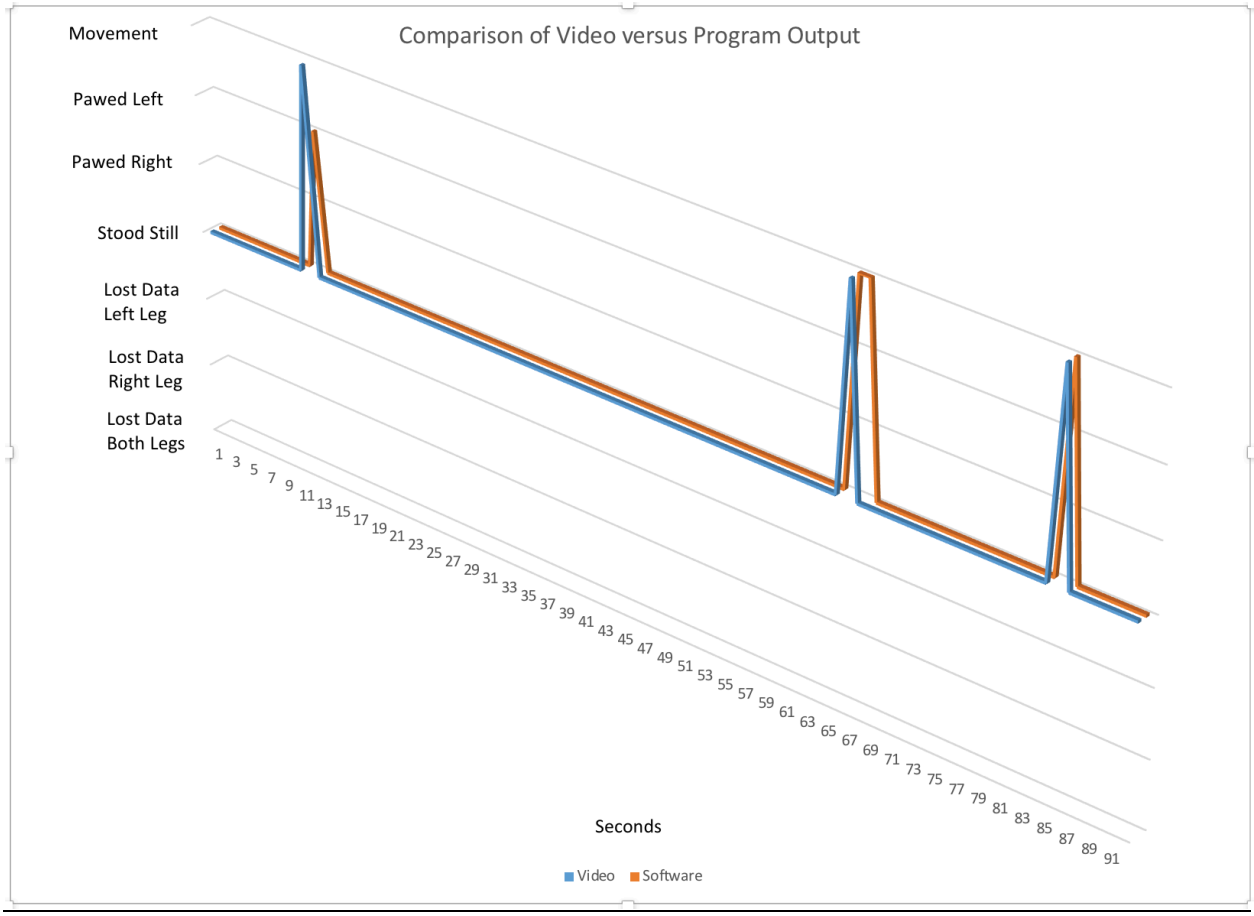

Figure 4.7: Results of One Second Synopsis – Slow Moving Equine

The detected behavior in 91 of the 92 possible intervals matched the observed behavior, yielding a 1.1% error rate. The one exception arose when the software mistakenly diagnosed movement as pawing (second 11 of Figure 4.7). This is due to how the algorithm detects pawing. If the equine takes a very slow step and its leg is suspended above the ground when an observation is made, the program categorizes the action as pawing.

The image processing latency issue noted above became evident at between time intervals 65 and 66. The equine's movement ended on the boundary between 65 seconds and 66 seconds,

whereas the program categorized the movement occurring over both 65 seconds and 66 seconds. We felt this was a minor issue, one that did not merit being categorized as a misclassification. Even counting it as a mismatch, the error rate increases slightly to 2.2%. Table 4.4 shows the one misclassification.

Type of Mismatch (video/program)	Number of Occurrences
Movement/Lost Data (all types)	0
Pawing/Movement	0
Data Loss Type Mismatch	0
Pawing/Standing Still	0
Movement/Pawing	1
Standing Still/Movement	0

Table 4.4: Mismatch Categories for One Second Analysis – Slow Moving Equine

Two Second Synopsis – Fast Moving Equine

Configuring the tracking and detection algorithm to use two seconds of data for the equine that moves rapidly in the stall (as opposed to one second) produced the results visualized in Figure 4.8.

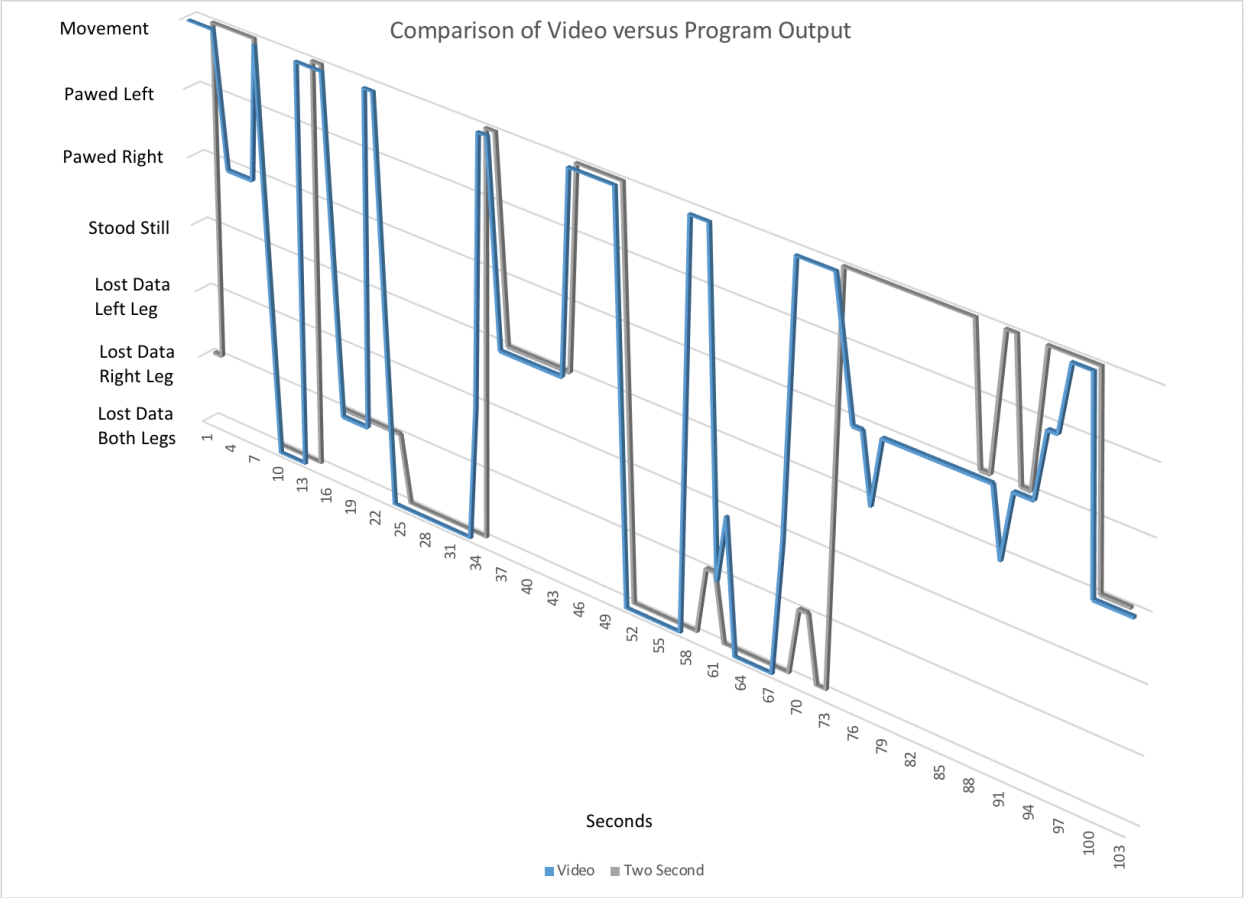

Figure 4.8: Results of Two Second Synopsis – Fast Moving Equine

The right leg pawing detection rate dramatically decreased when taking the extra second's worth of information into consideration. It went from 11 detections on the one second algorithm to 4 detections on the two second algorithm. The two-second algorithm did not detect the left leg pawing at all compared with the 3 detections of the one-second algorithm. The error rate jumped from 24.3% to 40.2%. Embracing the data in the additional second corrected the one false detection of the equine taking a step and the software mis-categorizing it as pawing.

Since the reconfigured algorithm required two seconds of data to complete, the number of comparison points decreased from 103 to 102. The observed versus detected behavior failed to

match 41 times, yielding an error rate of 40.2%. The mismatches fall into broad categories, as shown in Table 4.5.

Type of Mismatch (video/program)	Number of Occurrences
Movement/Lost Data (all types)	13
Pawing/Movement	21
Data Loss Type Mismatch	5
Pawing/Standing Still	0
Movement/Pawing	0
Standing Still/Movement	2

Table 4.5: Mismatch Categories for Two Second Analysis – Fast Moving Equine

Two Second Synopsis – Slow Moving Equine

Configuring the tracking and detection algorithm to use two seconds of data for the equine that moves very little in the stall (as opposed to one second) produced the results visualized in Figure 4.9.

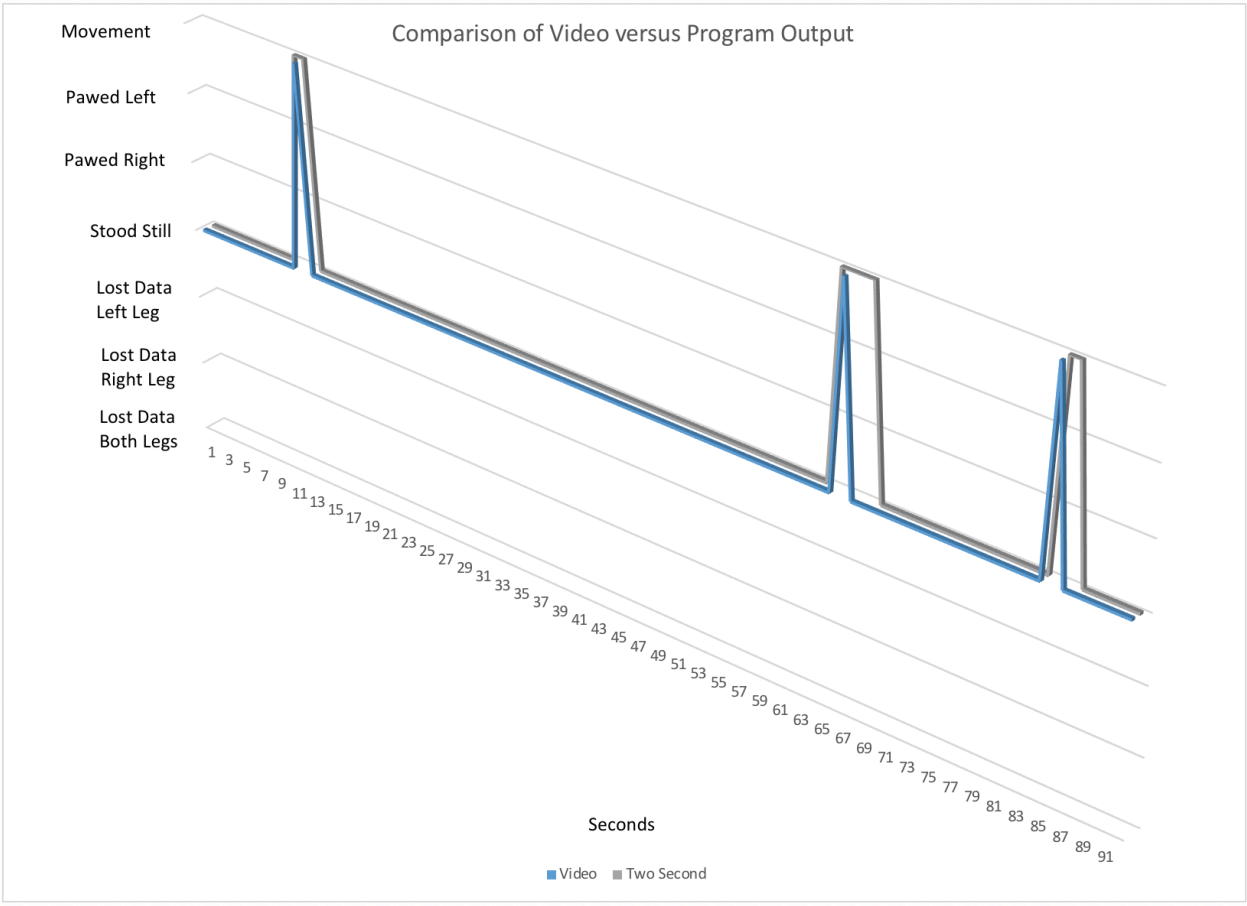

Figure 4.9: Results of Two Second Synopsis – Slow Moving Equine

The two-second algorithm remedied the single categorization mismatch of the one-second algorithm at the expense of introducing enough processing latency to mistakenly diagnose five comparison intervals. In all five instances, the software diagnosed the test subject as moving when it was observed to be standing still; this is due to the inherent processing latency in the software. If the equine's movement stopped within the first few frames of the two seconds the software would categorize the entire two seconds as movement; however to the observer the equine was standing still.

Counting the five discrepancies as diagnosis failures presents an error rate of 5.4%. The five mismatches fall into broad categories, as shown in Table 4.6.

Type of Mismatch (video/program)	Number of Occurrences
Movement/Lost Data (all types)	0
Pawing/Movement	0
Data Loss Type Mismatch	0
Pawing/Standing Still	0
Movement/Pawing	0
Standing Still/Movement	5

Table 4.6: Mismatch Categories for Two Second Synopsis – Slow Moving Equine

SYNOPSIS OF DATA – THIRD EQUINE

To ensure the algorithms weren't tuned for a specific equine, the research included a third equine in the analysis– one that was not used in any previous analyses or testing of the software. This equine also exhibits pawing behavior. This equine's activity level in the stall is medium – it is neither as fast as the fast moving equine's activity above nor as slow as the slow moving equine.

One Second Synopsis – Third Equine

Configuring the tracking and detection algorithm to use one second of data for the third equine produced the results visualized in Figure 4.10.

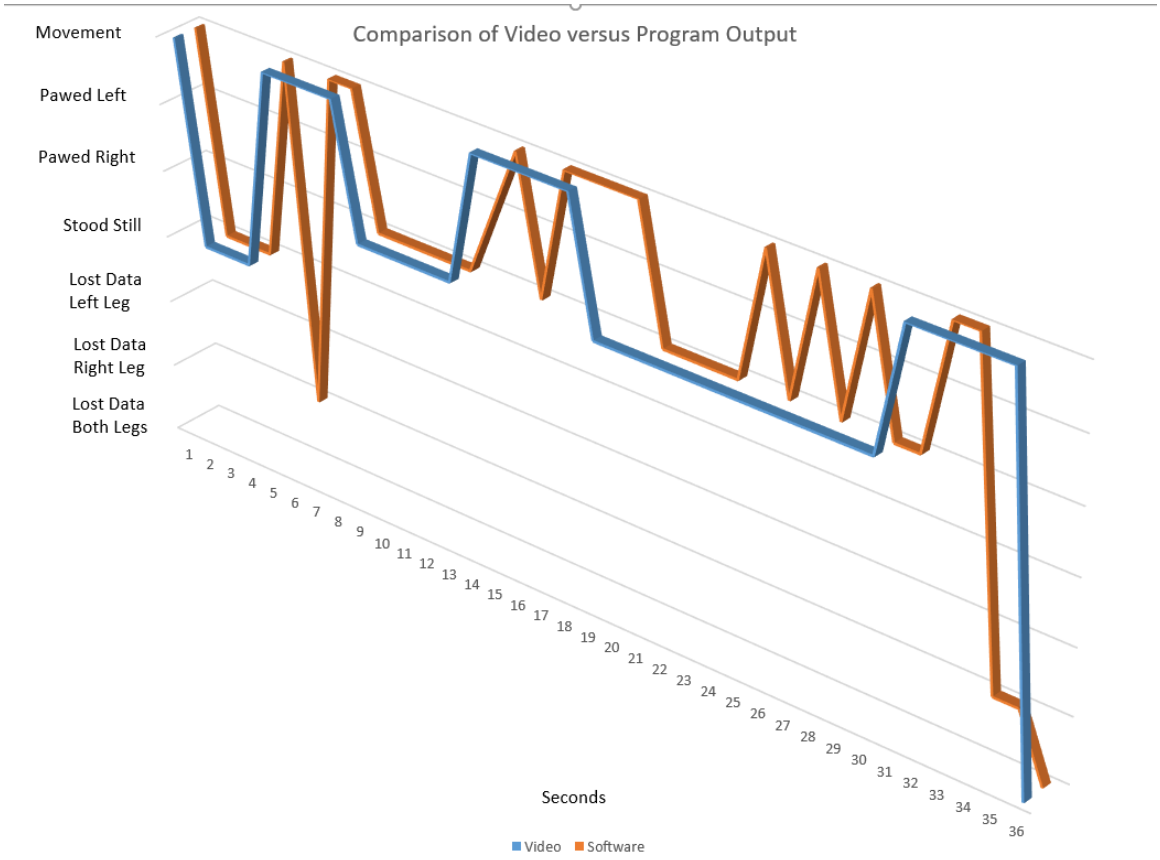

Figure 4.10: Results of One Second Synopsis – Third Equine

Eleven discrepancies were detected across the 36 total possible points for comparing observed and detected behavior, yielding a 30.6% error rate. These mismatches fall into broad categories as show in table 4.6. Analysis pointed to the following discrepancies between the observed and detected behaviors:

1. The detection mechanism lost track of the test subject's orientation in 3 of the 36 observation intervals.

2. The observed behavior was not accurately determined in 5 of the 36 cases. Ten of the faults diagnosed the test subject as moving when it was observed to be pawing. In the

remaining two faults, the test subject was observed to be pawing and was identified as doing something else.

Type of Mismatch (video/program)	Number of Occurrences
Movement/Lost Data (all types)	3
Pawing/Movement	5
Data Loss Type Mismatch	0
Pawing/Standing Still	0
Movement/Pawing	3
Standing Still/Movement	0

Table 4.7: Mismatch Categories for One Second Analysis – Third Equine

This equine twice demonstrates the same motion as pawing but never makes contact with the ground. The software diagnosed the equine as pawing but technically its movement. Even the observer had a hard time determining whether to classify it as pawing or movement. If these two mismatches are removed, the error rate decreases to 25%.

Two Second Analysis – Third Equine

Configuring the tracking and detection algorithm to use two seconds of data for the third equine produced the results visualized in Figure 4.11.

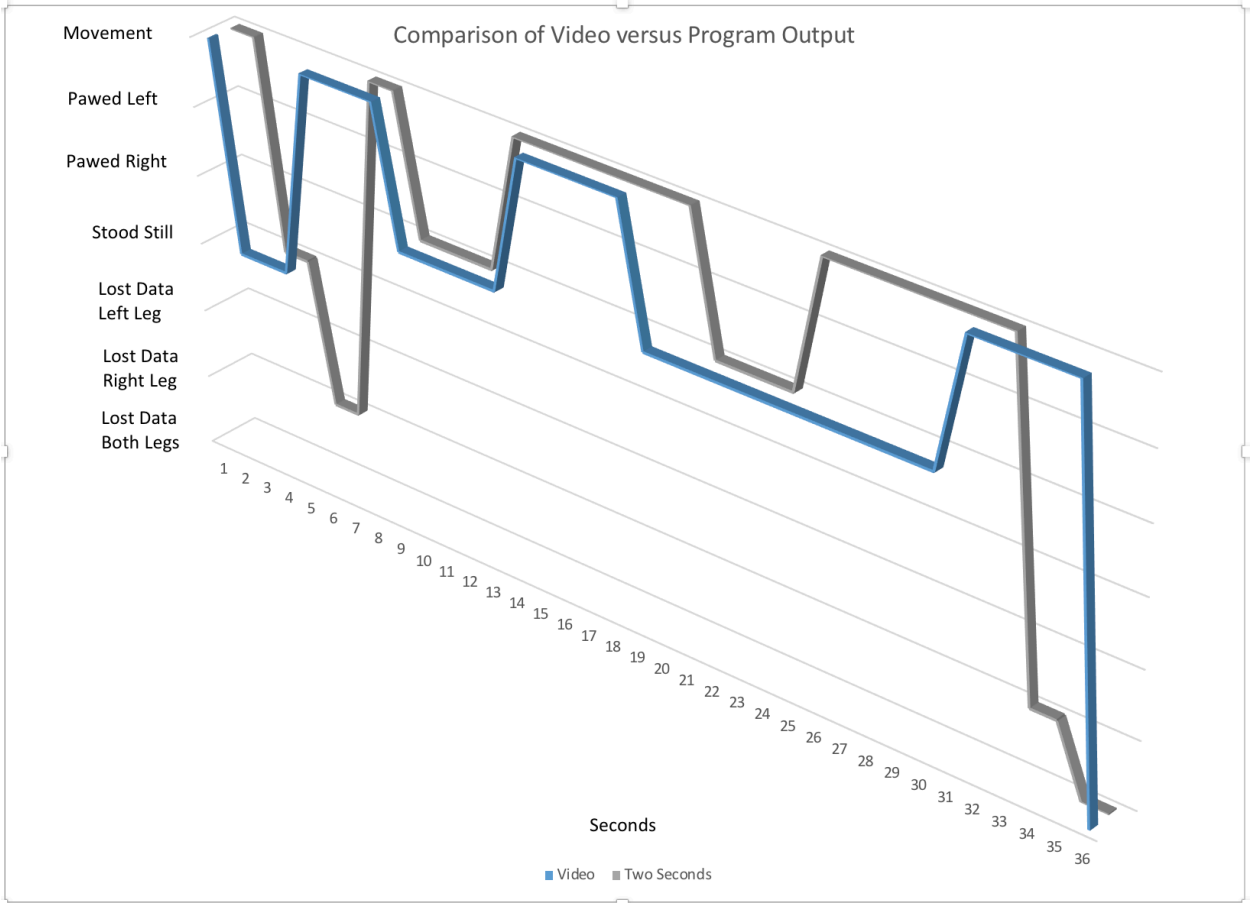

Figure 4.11: Results of Two Second Analysis – Third Equine

The detected behavior in 15 of the 36 possible intervals matched the observed behavior, yielding a 41.7% error rate. The motion erroneously detected as pawing with the one second algorithm was corrected with the two second algorithm. However this was at the cost of the algorithm missing four additional pawing detections.

The 15 mismatches fall into broad categories, as shown in Table 4.8

Type of Mismatch (video/program)	Number of Occurrences
Movement/Lost Data (all types)	5
Pawing/Movement	9
Data Loss Type Mismatch	0
Pawing/Standing Still	0
Movement/Pawing	0
Standing Still/Movement	1

Table 4.8: Mismatch Categories for Two Second Analysis – Third Equine

DATA ANALYSIS PHASE

Table 4.9 shows the error rates for each of the algorithms.

Equine	One Second Algorithm	Two Second Algorithm
Fast	24.3%	40.2%
Slow	1.1%	5.4%
Third	25.0%	41.7%

Table 4.9: Error Rates for Algorithms

Table 4.10 shows the percentage of mismatch types for each type of algorithm.

Type of Mismatch (video/program)	One Second Fast	One Second Slow	One Second Third	Two Second Fast	Two Second Slow	Two Second Third
Movement/Lost Data (all types)	44%	0%	27%	32%	0%	33%
Pawing/Movement	40%	0%	46%	51%	0%	60%
Data Loss Type Mismatch	8%	0%	0%	12%	0%	0%
Pawing/Standing Still	4%	0%	0%	0%	0%	0%
Movement/Pawing	4%	100%	27%	0%	0%	0%
Standing Still/Movement	0%	0%	0%	5%	100%	7%

Table 4.10: Percentage Mismatches

The software currently collects data at 2 Hz and makes a behavior determination every second. A rapidly moving equine can perform several behaviors in this half second lag between frames – all of which are not captured by the software. The observer does not have this problem as the observer is watching a video taken at 30Hz. There are two main challenges faced by the software: Occlusions and Processing Lag.

Occlusions

An occlusion occurs when something blocks one or both of the legs from the view of the camera at the moment the software collected data. This type of data mismatch accounted for the highest mis-categorization for the fast moving equine and the second highest mis-categorization for the third equine. Of the two observations the software used to make a behavior determination, if one of the frames included an occlusion then the software could not determine what behavior the equine was doing during that second and categorized the behavior as "Lost Data". However, the observer could see in the video the equine was moving as the leg came back into view. For example, Figure 4.12 shows a common example of an occlusion. The body of the equine blocked the right bandage from the camera's view. Figure 4.13 was from three frames earlier in the video and shows the right bandage clearly. The software classifies this as lost data while the observer classifies it at movement.

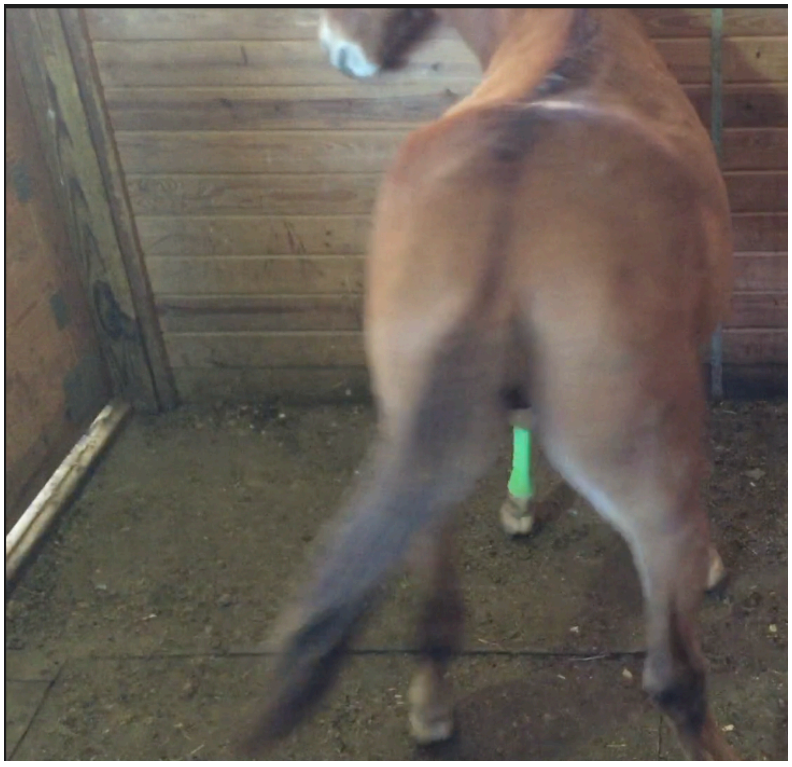

Figure 4.12: Example of an Occlusion

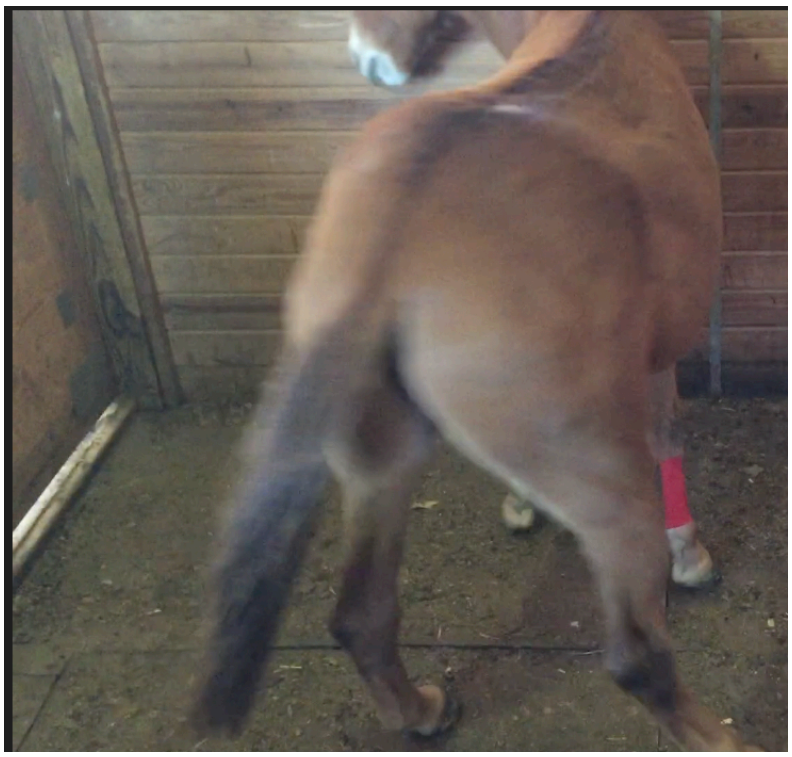

Figure 4.13: Three Video Frames Earlier than the Occlusion

Processing Lag

The software compares the movement of each individual leg from the previous frame and encodes this data into a 1 or 0. 1 represents movement was detected from the previous frame while a 0 represents no movement was detected.

When an equine paws, it exhibits a unique movement pattern. The non-pawing leg (aka the standing leg) stays still while the pawing leg moves rapidly back and forth. For one second's worth of data this can easily be encoded as 1010.

For the software to correctly match the pawing pattern, the software first has to collect the correct frames. This is where processing lag causes mismatches. For example, figures 4.14 and 4.15, represent two back-to-back frames (9.jpg and 10.jpg) collected by the software. As can be seen the left leg (green bandage) stayed still while the right leg (red bandage) moved. The encoded data is 1010 which the software correctly matched to pawing.

Figures 4.16 and 4.17 show where processing lag comes into play. These two figures are the next consecutive frames (11.jpg and 12.jpg). The movement pattern for this second of data is 1000. The right leg moves from frames 10.jpg and 11.jpg and is correctly encoded as 10; however, the leg doesn't appear to move between 11.jpg and 12.jpg and is encoded as 00. This creates a movement pattern for this second of data as 1000. Since movement occurred at some point during the interval (as represented by the 1) but does not match the pawing pattern, the software categorizes it as movement. This is due to the processing lag and the way Apple's AVFoundation manages photos. The method used to take the photo is an asynchronous process that "returns immediately after it is invoked, later calling the provided completion handler block

when image data is ready" and the developer "should not assume that the completion handler will be called on a specific thread" (Apple, 2018a, para. 5-6). This translates to a system lag if image processing does not complete in half a second because the buffer is not updated.

Figure 4.14: 9.jpg

Figure 4.15: 10.jpg

Figure 4.16: 11.jpg

Figure 4.17: 12.jpg

SUMMARY

This validation effort confirmed the positive hypothesis: “Smartphones can detect an equine pawing in a stall and can distinguish the pawing from an equine walking around in a stall.” There were challenges encountered such as lighting, occlusions, and processor lag. The lighting challenge was the easiest to overcome by performing a manual calibration for each run. Although nothing could completely eliminate the processor lag and occlusions, the software was still able to achieve a 75% - 98.9% success rate for the one second algorithm.

CHAPTER FIVE: SUMMARY

The overarching goal of this research was to use automated techniques to detect anomalous behavior exhibited by animals being observed in a controlled setting. The scholarly contribution of the work was in extending game-based modeling to describe patterns of behavior that are considered normal, to determine when observed behavior falls outside those patterns, and to diagnose the possible cause of the anomaly. The application domain of this endeavor was equine health. The research resulted in a proof-of-feasibility system that uses an off-the-shelf mobile phone to observe an equine while in a stall and detect when the equine is pawing.

The research was conducted in three main phases:

1. Modeling Phase
2. Data Acquisition/Behavior Recognition Phase
3. Analysis Phase

In the modeling phase, we developed a pattern-based model for describing a specific subset of equine behavior that can be delineated using the locomotion of the front two legs. During the data acquisition/behavior recognition phase, computer vision techniques were used to process the images in real-time to remove noise and extract the desired features from the images. Next, we used a game-based rule system to determine if the extracted features relocated from the last known position. The analysis phase used pattern matching to compare the a priori models developed in the modeling phase to the observed events from the data acquisition/behavior recognition phase to diagnosis the equine' behavior.

The positive hypothesis "H1 – Smartphones can detect an equine pawing in a stall and can distinguish the pawing from an equine walking around or standing still in a stall" was confirmed

through this research. A single unmodified smartphone (an iPhone 6) was used to detect abnormal behavior with an accuracy of 70-98% depending on the rapidity of the test subject's movement.

CONCLUSIONS

Initially, we attempted to track the equine's legs without the use of markers. However, there were many challenges to this approach as an equine's left leg and right leg look very similar unless the equine has a unique color pattern on one leg versus another. Complex algorithms would be necessary to distinguish one front leg from another, and these algorithms must be tailored to the individual horse. The algorithms would also have to account for occlusions and cross-overs. Currently such algorithms require high-end workstations; contemporary smartphones lack the processing power. For example, Vicon's Tracker System requires a minimum of a 3.6GHz processor along with a separate 2GB graphics card (Vicon, 2018). The entry-level software package by iPiSoft, which bills itself as Motion Capture for the Masses requires a minimum of a 2GHz processor along with a separate gaming-class graphics video card with the caveats that this "is an entry-level configuration that works with one depth sensor (Kinect 2 or Kinect). Easy to use but applicable for relatively simple motions without rotations" (Ipisoft, 2018, para. 1). In comparison the iPhone 6 has a 1.4Ghz processor with an integrated GPU (TechReport, 2018).

Smartphones can detect abnormal behavior using marker-based computer vision techniques albeit with less than 100% accuracy. To increase accuracy with the current marker-based approach the ability to process higher framerates (60+ fps) in real-time would be required.

However, storage capacity and energy consumption must be considered as the processing rate is increased. For example, in this research, an average picture size was 100KB. At a 2HZ rate, this meant 200KB was used every minute. At 60 HZ, 6MB each minute would be required for storage capacity. Similarly, the energy consumption would need to be monitored closely. At the highest processing peak the program reached 76% CPU utilization, quickly draining the battery.

The lowest resolution setting available on the iPhone 6 is 640x480 (Apple, 2018c). At this resolution the camera was able to detect the markers at a distance of 12 feet. This particular application required neither a higher resolution nor increased processing latency and storage space to achieve acceptable results.

Three separate test subjects with varying degrees of rapidness of movement were used for the data analysis portion with two different algorithms run against each. The first algorithm analyzed one second's data of movement before determining the behavior being exhibited. The second algorithm used two seconds' worth of data. The first test subjected moved very rapidly and the error rates were 24.3% for the first algorithm and 40.2% for the second algorithm. The second test subject moved very little in the stall and the error rates were 1.1% for the first algorithm and 5.4% for the second algorithm. The third test subject had a medium movement rate and the error rates were 30.6% for the first algorithm and 41.7% for the second algorithm. The number of observations made per second and the number of consecutive observations considered affect the accuracy of the diagnosis. Increasing either, or both, improves fidelity and provides a sounder basis for diagnosis. This is done at the cost of increasing the amount of computing required to process the data.

Apple's gaming framework, GameplayKit, provided several algorithms and data structures useful for building computer vision applications. First, GameplayKit provided the ability to easily build a model where the equine is separated from its associated behaviors. The equine was represented as an entity and the behaviors as components with each behavior being segmented into a single component. This segmentation made the model extensible where behaviors can easily be added without effecting the current existing behaviors. The framework takes these rules and determines what behavior is being displayed based on what rules are asserted from the current state of the system. Using a rule-based system allows for easier maintainability by allowing rules to be easily added, deleted or modified.

FUTURE WORK

Although this endeavor successfully demonstrated the feasibility of monitoring equine behavior, it also illustrated the complexity of doing so. Additional work required to mature this research to the level of a useful product include the following:

- Port software to other platforms. Currently the software is specifically tied to the Apple line of smartphones since it uses Apple's Gameplay Kit for implementation.
- Extend Equine Behavior Indicators. The framework was purposely architected so that the detection mechanism was decoupled from the model description. This allows behaviors to be added without having to modify the software that detects those behaviors. Candidate equine behaviors for inclusion are foundering, rolling, and kicking.
- Extend behaviors beyond the equine domain. This research focused on detection of equine behavior. The same modeling concepts can be used for other animals.

- Build a dynamic behavioral model. Build a model of the behavior of the test subject "on-the-fly" and use this model as the template for normal behavior. Flag behavior that falls outside this template as abnormal.
- Add additional sensors. Add additional sensor inputs to the framework to detect even more behavior. For example, accelerometers could be added to detect how quickly the animal is moving.
- Adjust timing of observations. Currently, the software takes new observations at preset intervals. Adjust timing of capturing new frames based on the quickness of the movement of the animal.
- Use multiple smartphones. Use multiple smartphones placed at different locations to capture a larger field of view.
- Automatically adapt to lighting conditions. Develop an algorithm that would automatically adjust to the current lighting conditions.
- Use active markers. The current approach uses markers that requires enough light to be present for the camera to see the markers. To adapt this to nighttime use, use markers that actively emit light.

Much of the future work discussed above will require more processing power than smartphones currently possess. However, according to Experts Exchange, the processing power of smartphones have increased 1 trillion-fold in the past 60 years(Experts-Exchange, 2018). If this trend continues, the accuracy of detection could be increased as more sophisticated algorithms could be developed and utilized on a smartphone.

REFERENCES

Abbas, S.M. and Muhammad, A. (2012). Outdoor RGB-D SLAM Performance in Slow Mine Detection. *Proceedings of ROBOTIK 2012; 7th German Conference on Robotics*. 1-6.

Abson, K., & Palmer, I. (2015). Motion capture: capturing interaction between human and animal. *The Visual Computer*. Vol 31(3), 341-353.

Agyemang, M., Barker, K., & Alhajj, R. (2006). A comprehensive survey of numeric and symbolic outlier mining techniques. *Intelligent Data Analysis*. Vol 10(6), 521-538.

American Horse Council. (2005). National Economic Impact of the U.S. Horse Industry. *AmericanEquestrian.com*. Retrieved 3 October 2018, from http://www.americanequestrian.com/pdf/American_Horse_Council_2005_Report.pdf

Apple. (2016). iPhone 6 - Technical Specifications - Apple. *Apple.com*. Retrieved 6 August 2016, from http://www.apple.com/iphone-6/specs/

Apple. (2018a). captureStillImageAsynchronouslyFromConnection:completionHandler *Apple.com*. Retrieved 4 October 2018 https://developer.apple.com/documentation/avfoundation/avcapturestillimageoutput/1387374-capturestillimageasynchronouslyf

Apple. (2018b). Gameplay Kit. *Developer.apple.com.* Retrieved 14 October 2018, from https://developer.apple.com/documentation/gameplaykit

Apple. (2018c). Still and Video Media Capture - Apple. *Apple.com*. Retrieved 22 September 2018, from https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/AVFoundationPG/Articles/04_MediaCapture.html

Amazon. (2016). iKross Universal Compact Flexible Tripod Stand Holder with Adapters For smartphone / Digital Camera / GoPro Hero All Version: Cell Phones & Accessories. *Amzn.com*. Retrieved 6 August 2016, from http://amzn.com/B00T8VXYAU

Anguita, D., Ghio, A., Oneto, L., Parra, X., & Reyes-Ortiz, J. L. (2012). Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. *Ambient assisted living and home care*. 216-223. Springer Berlin Heidelberg.

Ayusawa, K., Ikegami, Y., & Nakamura, Y. (2014). Simultaneous global inverse kinematics and geometric parameter identification of human skeletal model from motion capture data. *Mechanism and Machine Theory*. Vol 74, 274-284.

Back, W., & Clayton, H. M. (2013). *Equine locomotion*. Elsevier Health Sciences.

Barca, J. C., Rumantir, G., & Li, R. K. (2006). A new illuminated contour-based marker system for optical motion capture. *Innovations in Information Technology, 2006*. 1-5. IEEE.

Beatty, M. F. (1986). *Principles of Engineering Mechanics: Volume 1 Kinematics - The geometry of motion*. Vol. 32. Springer.

Bregler, C. (2007). Motion capture technology for entertainment [in the spotlight]. *Signal Processing Magazine*. Vol 24(6), 156-160. IEEE

Butler, C. & Houpt, K. (2014). Pawing by Standardbred Racehorses: Frequency and Patterns. *Journal of Equine Science,* Vol 25(3), 57-59.

Cha, G., Gong, J., & Oh, S. (2014, August). SmartPTA: A smartphone-Based Human Motion Evaluation System. *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2014 IEEE International Conference*. 43-48. IEEE.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*. Vol 41(3), 15.

Colorado State University. (2014). Colic Surgery Cost Estimates. *Csu-cvmbs.colostate.edu*. Retrieved 5 August 2016, from http://csu-cvmbs.colostate.edu/Documents/equine-medicine-surgery-colic-cost-estimates.pdf

Conklin, E. E., Lee, K. L., Schlabach, S. A., & Woods, I. G. (2015). VideoHacking: Automated Tracking and Quantification of Locomotor Behavior with Open Source Software and Off-the-Shelf Video Equipment. *Journal of Undergraduate Neuroscience Education*. Vol 13(3), A120-A125.

Corazza, S., Muendermann, L., Chaudhari, A. M., Demattio, T., Cobelli, C., & Andriacchi, T. P. (2006). A markerless motion capture system to study musculoskeletal biomechanics: visual hull and simulated annealing approach. *Annals of biomedical engineering*. Vol 34(6), 1019-1029.

CrunchBase. (2016). Ubersense | CrunchBase. *Crunchbase.com*. Retrieved 6 May 2016, from https://www.crunchbase.com/organization/ubersense#/entity

De Aguiar, E., Theobalt, C., Stoll, C., & Seidel, H. P. (2007). Marker-less deformable mesh tracking for human shape and motion capture. *Computer Vision and Pattern Recognition, 2007*. 1-8. IEEE.

Delorme, S., Lamontagne, M., & Tavoularis, S. (2002). Kinematic measurements of snowboarder's ankles. *World Congress on BioMech*. Calgary, Canada.

Dernbach, S., Das, B., Krishnan, N. C., Thomas, B. L., & Cook, D. J. (2012). Simple and complex activity recognition through smart phones. *Intelligent Environments (IE), 2012 8th International Conference*. 214-221. IEEE.

Dutta, T. (2012). Evaluation of the Kinect™ sensor for 3-D kinematic measurement in the workplace. *Applied ergonomics*. Vol 43(4), 645-649.

Entity Systems Wiki. (2014). What's an Entity System?. *Entity-systems-wiki.t-machine.org*. Retrieved 14 October 2018, from http://entity-systems-wiki.t-machine.org

Experts-Exchange. (2018). Processing Power Compared. *Pages.Experts-Exchange.Com*. Retrieved 22 September 2018, from https://pages.experts-exchange.com/processing-power-compared

Étienne-Jules_Marey. (2016). *Wikipedia.org*. Retrieved 7 August 2016, from https://en.wikipedia.org/wiki/Étienne-Jules_Marey

Garrido, J. E., Marset, I., Penichet, V. M., & Lozano, M. D. (2013, May). Balance disorder rehabilitation through movement interaction. *Proceedings of the 7th International Conference on Pervasive Computing Technologies for Healthcare*. 319-322. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Gogoi, P., Bhattacharyya, D. K., Borah, B., & Kalita, J. K. (2011). A survey of outlier detection methods in network anomaly identification. *The Computer Journal*.

Google. (2018). Search for "Sensor Definition". *Google.com*. Retrieved 14 October 2018, from https://www.google.com

Győrbíró, N., Fábián, Á., & Hományi, G. (2009). An activity recognition system for mobile phones. *Mobile Networks and Applications*. Vol 14(1), 82-91.

He, Y., & Li, Y. (2013). Physical activity recognition utilizing the built-in kinematic sensors of a smartphone. *International Journal of Distributed Sensor Networks, 2013*.

Hen, Y. W., & Paramesran, R. (2009, December). Single camera 3d human pose estimation: A review of current techniques. *Technical Postgraduates (TECHPOS), 2009 International Conference*. 1-8. IEEE.

Hilo, U. (2016). Add-on lens for the iPhone camera, gives you the right angle. *Shop.hilolens.com*. Retrieved 2 May 2016, from http://shop.hilolens.com/?menu

Hodge, V. J., & Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*. Vol 22(2), 85-126.

House, Michael. (2012). What is the role of "systems" in a component-based entity architecture?. *Gamedev.stackexchange.com*. Retrieved 14 October 2018, from https://gamedev.stackexchange.com/questions/31473/what-is-the-role-of-systems-in-a-component-based-entity-architecture

Howe, N. R., Leventon, M. E., & Freeman, W. T. (1999). Bayesian Reconstruction of 3D Human Motion from Single-Camera Video. NIPS. Vol 99. 820-6.

iPhone 6. (2016). *Wikipedia.org*. Retrieved 6 August 2016, from https://en.wikipedia.org/wiki/IPhone_6

Ipisoft. (2018). *Ipisoft.com*. Retrieved 26 September 2018, from http://ipisoft.com/software/express-edition/

Kapsouras, I., & Nikolaidis, N. (2014). Action recognition on motion capture data using a dynemes and forward differences representation. *Journal of Visual Communication and Image Representation*. Vol 25(6), 1432-1445.

Khan, A. M., Lee, Y. K., Lee, S. Y., & Kim, T. S. (2010). Human activity recognition via an accelerometer-enabled-smartphone using kernel discriminant analysis. *Future Information Technology (FutureTech), 2010 5th International Conference*. 1-6. IEEE.

Kim, A., Kim, J., Rietdyk, S., & Ziaie, B. (2015). A wearable smartphone-enabled camera-based system for gait assessment. *Gait & posture*. Vol 42(2), 138-144.

Kinetisense. (2018). Kinetisense User Manual. *Kinetisense.com*. Retrieved 7 November 2018, from https://kinetisense.com/kinetisense-user-manual-4/#cat-1

Krayevoy, V., & Sheffer, A. (2005). Boneless motion reconstruction. *ACM SIGGRAPH 2005 Sketches*. 122. ACM.

Ku, K. (2014). Equine body weight estimation using three-dimensional images. Masters Thesis. *Colorado State University*. Retrieved 25 April 2016, from https://dspace.library.colostate.edu/handle/10217/167238

Kumar, N., Kunju, N., Kumar, A., & Sohi, B. S. (2010). Active marker based kinematic and spatio-temporal gait measurement system using LabVIEW vision. *Journal of Scientific and Industrial Research*. Vol 69, 600-605.

Lee, S. J., Tewolde, G., Lim, J., & Kwon, J. (2015, July). QR-code based Localization for Indoor Mobile Robot with validation using a 3D optical tracking instrument. *Advanced Intelligent Mechatronics (AIM), 2015 IEEE International Conference*. 965-970. IEEE.

Lengyel, László. (2015). Validating rule-based algorithms. *Acta Polytechnica Hungarica* 12.4. Retrieved 14 October 2018, from https://www.uni-obuda.hu/journal/Lengyel_60.pdf

Melexis. (2016). Melexis: Hall-effect Position Sensors | Sensorless BLDC Motor Drivers. *Melexis.com*. Retrieved 23 April 2016, from http://www.melexis.com/Assets/What-is-an-optical-sensor-3913.aspx

Meyer, J., Kuderer, M., Muller, J., & Burgard, W. (2014). Online marker labeling for fully automatic skeleton tracking in optical motion capture. *Robotics and Automation (ICRA), 2014 IEEE International Conference*. 5652-5657. IEEE.

MI-AS. (2016). Motion Analysis Software for Biological and Veterinary Sciences (Equine and Canine Gait). *MI-AS.com*. Retrieved 2 May 2016, from http://www.mi-as.com/applications/biological-and-veterinary-sciences/

Miller, G. (2012). The smartphone psychology manifesto. *Perspectives on Psychological Science*. Vol 7(3), 221-237.

Milne, A. D., Chess, D. G., Johnson, J. A., & King, G. J. W. (1996). Accuracy of an electromagnetic tracking device: a study of the optimal operating range and metal interference. *Journal of biomechanics*. Vol 29(6), 791-793.

Moeslund, T. B., & Granum, E. (2001). A survey of computer vision-based human motion capture. *Computer vision and image understanding*. Vol 81(3), 231-268.

Moeslund, T. B., Hilton, A., & Krüger, V. (2006). A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding*. *Vol* 104(2), 90-126.

Nie, Y., Ishii, I., Yamamoto, K., Orito, K., & Matsuda, H. (2009). Real-time scratching behavior quantification system for laboratory mice using high-speed vision. *Journal of real-time image processing*. Vol 4(2), 181-190.

Olsen, E., Haubro Andersen, P., & Pfau, T. (2012). Accuracy and precision of equine gait event detection during walking with limb and trunk mounted inertial sensors. *Sensors*. Vol 12(6), 8145-8156.

Okada, R., & Stenger, B. (2008). A single camera motion capture system for human-computer interaction. *IEICE TRANSACTIONS on Information and Systems*. Vol 91(7), 1855-1862.

Ouchi, K., & Doi, M. (2012). Indoor-outdoor activity recognition by a smartphone. *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. 600-601. ACM.

Pannell, G., & Ashman, H. (2010, June). User modelling for exclusion and anomaly detection: a behavioural intrusion detection system. *International Conference on User Modeling, Adaptation, and Personalization*. 207-218. Springer Berlin Heidelberg.

PCMag. (2018). Definition of: application framework. *PCMag.com*. Retrieved 14 October 2018, from https://www.pcmag.com/encyclopedia/term/37907/application-framework

Penn. (2012). Penn Vet's Farrier Service Keeps Horses on Their Feet. *Upenn.edu*. Retrieved 23 April 2016, from http://www.upenn.edu/pennnews/news/penn-vet-s-farrier-service-keeps-horses-their-feet

Persistence. (2015). Non-Optical Semiconductor Sensor Market: Global Industry Analysis and Forecast 2015 - 2021. *Persistencemarketresearch.com*. Retrieved 22 April 2016, from http://www.persistencemarketresearch.com/market-research/non-optical-semiconductor-sensor-market.asp

Pittman, J. T., & Ichikawa, K. M. (2013). iPhone® applications as versatile video tracking tools to analyze behavior in zebrafish (Danio rerio). *Pharmacology Biochemistry and Behavior*. Vol 106, 137-142.

Polar. (2016). Equine | Polar USA. *Polar.com*. Retrieved 23 April 2016, from http://www.polar.com/us-en/products/equine/

Polhemus. (2016). Polhemus is the premier precision motion tracking company. *Polhemus.com*. Retrieved 22 April 2016, from http://polhemus.com/motion-tracking/all-trackers/fastrak

Poppe, R. (2007). Vision-based human motion analysis: An overview. *Computer vision and image understanding*. Vol 108(1), 4-18.

Pueo, Basilio. (2016). "High speed cameras for motion analysis in sports science." Journal of Human Sport and Exercise 11.1 pp 53-73. Retrieved 14 October 2018, from http://www.redalyc.org/pdf/3010/301049620005.pdf

Qualisys. (2016). Qualisys | Motion Capture Systems. *Qualisys.com*. Retrieved 2 May 2016, from http://www.qualisys.com

Qualisys-Picture. (2016). *Qualisys.com*. Retrieved 2 May 2016, from http://www.qualisys.com/applications/other/

Qualisys-PDF. (2012). Equine Kinematics for an objective analysis of horse movements. *Qualisys.com*. Retrieved 2 May 2016, from http://content.qualisys.com/2015/10/AN_Equine.pdf

Quaranta, A., Siniscalchi, M., & Vallortigara, G. (2007). Asymmetric tail-wagging responses by dogs to different emotive stimuli. *Current Biology*. 17(6), R199-R201.

Quesada, L., & León, A. J. (2011). The Object Projection Feature Estimation Problem in Unsupervised Markerless 3D Motion Tracking. *arXiv.org* Retrieved 2 May 2016, http://arxiv.org/abs/1111.3969

Quesada, L., & León, A. J. (2012). Filling-Based Techniques Applied to Object Projection Feature Estimation. *arXiv.org*. Retrieved 2 May 2016, http://arxiv.org/abs/1202.6586

Ramanan, D., & Baker, S. (2011). Local distance functions: A taxonomy, new algorithms, and an evaluation. *Pattern Analysis and Machine Intelligence, IEEE Transactions*. Vol 33(4), 794-806.

Rocca, F., De Deken, P. H., Grisard, F., Mancas, M., & Gosselin, B. Real-time marker-less implicit behavior tracking for user profiling in a TV context. Semanticscholoar.org. Retrieved 25 April 2016, from https://www.semanticscholar.org/paper/Real-time-Marker-less-Implicit-Behavior-Tracking-Rocca-Deken/c69bb5bc2dde9ce793b30dc47f4a8dee868cc023/pdf

Rosenhahn, B., Brox, T., Kersting, U., Smith, A., Gurney, J., & Klette, R. (2006). A system for marker-less motion capture. *Künstliche Intelligenz*. Vol (2006), 45-51.

Roshtkhari, M., & Levine, M. (2013). Online dominant and anomalous behavior detection in videos. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2611-2618.

Roters, J., Jiang, X., & Rothaus, K. (2011). Recognition of traffic lights in live video streams on mobile devices. *Circuits and Systems for Video Technology, IEEE Transactions*. Vol 21(10), 1497-1511.

Rougier, C., & Meunier, J. (2010). 3D head trajectory using a single camera. *Image and Signal Processing*. 505-512. Springer Berlin Heidelberg.

Schubert, T., Gkogkidis, A., Ball, T., & Burgard, W. (2015, May). Automatic initialization for skeleton tracking in optical motion capture. *Robotics and Automation (ICRA), 2015 IEEE International Conference*. 734-739. IEEE.

Schwendimann, Beat. (2010). What is the difference between a simulation and a model? [Updated]. *Proto-Knowledge.com*. Retrieved 14 October 2018, from https://proto-knowledge.blogspot.com/2010/12/what-is-difference-between-simulation.html?m=1

Shang, Y., Zeng, W., Ho, D. K., Wang, D., Wang, Q., Wang, Y., ... & Rui, L. (2012, January). Nest: Networked smartphones for target localization. *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*. 732-736. IEEE.

Smith, A. (2015). U.S. smartphone Use in 2015. *Pewinternet.org*. Retrieved 6 August 2016, from http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/

Smyth, T. T., Carmalt, J. L., Treen, T. T., & Lanovaz, J. L. (2015). The effect of acute unilateral inflammation of the equine temporomandibular joint on the kinematics of mastication. *Equine veterinary journal. Vol 48(4), 523-527. |https://doi.org/10.1111/evj.12452*

Song, X., Wu, M., Jermaine, C., & Ranka, S. (2007). Conditional anomaly detection. *Knowledge and Data Engineering*. Vol 19(5), 631-645. IEEE.

Sports Motion Analyzer. (2016). Sports Motion Analyzer on the App Store. *Itunes.apple.com*. Retrieved 8 August 2016, from https://itunes.apple.com/us/app/sports-motion-analyzer/id419261296?mt=8

Stauffer, C., & Grimson, W. E. L. (1999). Adaptive background mixture models for real-time tracking. *Computer Vision and Pattern Recognition, 1999*. Vol 2. IEEE.

Stavrakakis, S., Li, W., Guy, J. H., Morgan, G., Ushaw, G., Johnson, G. R., & Edwards, S. A. (2015). Validity of the Microsoft Kinect sensor for assessment of normal walking patterns in pigs. *Computers and Electronics in Agriculture*. Vol 117, 1-7.

Structure. (2016). Structure Sensor – Capture the World in 3D. *Structure.io*. Retrieved 26 April 2016, from https://store.structure.io/store

Takahashi, D. (2011). Organic Motion launches new platform to capture motion. *VentureBeat.com*. Retrieved 25 April 2016, from http://venturebeat.com/2011/08/09/organic-motion-launches-new-platform-to-capture-motion/

Tao, Y., Hu, H., & Zhou, H. (2007). Integration of vision and inertial sensors for 3D arm motion tracking in home-based rehabilitation. *The International Journal of Robotics Research*, Vol 26(6), 607-624.

Tariq, S., & Dellaert, F. (2004). A multi-camera 6-dof pose tracker. *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*. 296-297. IEEE Computer Society.

TechReport. (2018). *Techreport.com*. Retrieved 26 September 2018, from https://techreport.com/review/27376/apple-iphone-6-and-6-plus-smartphones-reviewed/3

Trochim, W. M. (1989). Outcome pattern matching and program theory. *Evaluation and program planning*, 12(4), 355-366. Retrieved 14 October 2018, from https://www.socialresearchmethods.net/research/Outcome%20Pattern%20Matching%20and%20Program%20Theory.pdf

UC Davis Center for Equine Health. (2008). Colic: An Age Old Problem. *CEH Horse Report*. Vol 26(1).

Vicon. (2016). *Vicon.com*. Retrieved 2 May 2016, from http://www.vicon.com/what-is-motion-capture

Vicon. (2018). *Vicon.com*. Retrieved 26 September 2018, https://www.vicon.com/faqs/operating-systems-and-pc-specification

Vondrak, M., Sigal, L., Hodgins, J., & Jenkins, O. (2012). Video-based 3D motion capture through biped control. *ACM Transactions On Graphics (TOG)*. Vol 31(4), 27.

Wang, Q., Lobzhanidze, A., Jang, H. I., Zeng, W., & Shang, Y. (2012, July). Video based real-world remote target tracking on smartphones. *Multimedia and Expo (ICME), 2012 IEEE International Conference*. 693-698. IEEE.

Weber, M., Alexander, T., & Amor, H. B. (2008). Enhancing Motion Capture Performance by Means of an Internal Anthropometric Skeleton Model (No. 2008-01-1927). *SAE Technical Paper*.

Wiles, A. D., Thompson, D. G., & Frantz, D. D. (2004). Accuracy assessment and interpretation for optical tracking systems. *Medical Imaging 2004*. 421-432. International Society for Optics and Photonics.

Xcitex. (2016). Xcitex Store Motion Analysis Software and Capture Systems. *Xcitex.com*. Retrieved 25 April 2016, from http://www.xcitex.com/xcitex-motion-analysis-software-data-video-synchronization-capture-store.php

Xia, L., Chen, C. C., & Aggarwal, J. K. (2011). Human detection using depth information by kinect. *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference*. 15-22. IEEE.

Xsens. (2016). World's first 3D inertial motion capture of a horse in motion - Xsens 3D motion tracking. *Xsens.com*. Retrieved 23 April 2016, from https://www.xsens.com/press-releases/worlds-first-3d-inertial-motion-capture-horse-motion/

Zanero, S. (2004). Behavioral intrusion detection. *Computer and Information Sciences-ISCIS 2004*. 657-666. Springer Berlin Heidelberg.

APPENDIX A: SOFTWARE CODE

```
//  PawingComponent.h
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import <Foundation/Foundation.h>

@interface PawingComponent : NSObject

- (void)pawing:(NSString *)whichLeg myFile:(NSFileHandle
*)myFile;

@end
```

```
//  PawingComponent.m
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import "PawingComponent.h"

@implementation PawingComponent

- (void)pawing:(NSString *)whichLeg myFile:(NSFileHandle
*)myFile{
    if ([whichLeg isEqualToString:@"Left Front"]){
        [myFile seekToEndOfFile];
        NSString *leftLeg = @"the horse pawed with the left
leg\n\n";
        [myFile writeData:[leftLeg
dataUsingEncoding:NSUTF8StringEncoding]];
    }else{
        [myFile seekToEndOfFile];
        NSString *rightLeg = @"the horse pawed with the right
leg\n\n";
        [myFile writeData:[rightLeg
dataUsingEncoding:NSUTF8StringEncoding]];
    }
}

@end
```

```
//  StandingStillComponent.h
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import <Foundation/Foundation.h>

@interface StandingStillComponent : NSObject

- (void)still:(NSFileHandle *)myFile;

@end
```

```
//  StandingStillComponent.m
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import "StandingStillComponent.h"

@implementation StandingStillComponent

- (void)still:(NSFileHandle *)myFile{
    [myFile seekToEndOfFile];
    NSString *standingStill = @"the horse stood still\n\n";
    [myFile writeData:[standingStill
dataUsingEncoding:NSUTF8StringEncoding]];
}

@end
```

```
//  MovingComponent.h
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import <Foundation/Foundation.h>

@interface MovingComponent : NSObject

- (void)moving:(NSFileHandle *)myFile;

@end
```

```
//  MovingComponent.m
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import "MovingComponent.h"

@implementation MovingComponent

- (void)moving:(NSFileHandle *)myFile{
    [myFile seekToEndOfFile];
    NSString *bothLegs = @"the horse moved\n\n";
    [myFile writeData:[bothLegs
dataUsingEncoding:NSUTF8StringEncoding]];
}

@end
```

```
//  HorseEntity.h
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import <Foundation/Foundation.h>
#import "PawingComponent.h"
#import "StandingStillComponent.h"
#import "MovingComponent.h"

@interface HorseEntity : NSObject

@property (readwrite, nonatomic) PawingComponent *pawing;
@property (readwrite, nonatomic) StandingStillComponent *still;
@property (readwrite, nonatomic) MovingComponent *moving;
@property (readwrite, nonatomic) unsigned int pawingLeftCounter;
@property (readwrite, nonatomic) unsigned int
pawingRightCounter;
@property (readwrite, nonatomic) unsigned int stillCounter;
@property (readwrite, nonatomic) unsigned int movingCounter;
@property (readwrite, nonatomic)NSArray *paths;
@property (readwrite, nonatomic)NSString *documentsDirectory;
@property (readwrite, nonatomic)NSString *documentTXTPath;

- (void)determineHorseBehavior:(NSMutableString *) movPattern;
- (void)printHorseStats:(NSString *) movPattern;

@end
```

```
//  HorseEntity.m
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import "HorseEntity.h"

@implementation HorseEntity


-(id)init {
    if ( self = [super init] ) {
        _pawing = [[PawingComponent alloc] init];
        _still = [[StandingStillComponent alloc] init];
        _moving = [[MovingComponent alloc] init];
        _pawingLeftCounter = 0;
        _pawingRightCounter = 0;
        _stillCounter = 0;
        _movingCounter = 0;

        self.paths =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,NSUserDo
mainMask, YES);
        self.documentsDirectory = [self.paths objectAtIndex:0];
        self.documentTXTPath = [self.documentsDirectory
stringByAppendingPathComponent:@"HorseStats.txt"];

        NSFileManager *fileManager = [NSFileManager
defaultManager];
        if(![fileManager fileExistsAtPath:self.documentTXTPath])
        {
            NSString *savedString = @"Horse
Stats\n*****START*****\n";
            [savedString writeToFile:self.documentTXTPath
atomically:YES];
        } else{
            NSFileHandle *myHandle = [NSFileHandle
fileHandleForWritingAtPath:self.documentTXTPath];
            [myHandle seekToEndOfFile];
            NSString *start = @"*****START*****\n";
            [myHandle writeData:[start
dataUsingEncoding:NSUTF8StringEncoding]];
        }
    }
    return self;
} //end init
```

```
- (void)determineHorseBehavior:(NSMutableString *) movPattern{
    NSFileHandle *myHandle = [NSFileHandle
fileHandleForWritingAtPath:self.documentTXTPath];
    [myHandle seekToEndOfFile];
    NSString *savedStringStars =
@"********************\nMovement Pattern: ";
    [myHandle writeData:[savedStringStars
dataUsingEncoding:NSUTF8StringEncoding]];
    [myHandle seekToEndOfFile];
    [myHandle writeData:[movPattern
dataUsingEncoding:NSUTF8StringEncoding]];
    [myHandle seekToEndOfFile];
    NSString *savedString = @"\nIn the last second: ";
    [myHandle writeData:[savedString
dataUsingEncoding:NSUTF8StringEncoding]];

    if ([movPattern rangeOfString:@"9"
options:NSRegularExpressionSearch].location != NSNotFound){
        [myHandle seekToEndOfFile];
        NSString *lostDataBoth = @"lost data on both legs\n\n";
        [myHandle writeData:[lostDataBoth
dataUsingEncoding:NSUTF8StringEncoding]];

    }  else if([movPattern rangeOfString:@"8"
options:NSRegularExpressionSearch].location != NSNotFound){
            [myHandle seekToEndOfFile];
            NSString *lostDataRight = @"lost data on the right
leg\n\n";
            [myHandle writeData:[lostDataRight
dataUsingEncoding:NSUTF8StringEncoding]];

    }  else if([movPattern rangeOfString:@"7"
options:NSRegularExpressionSearch].location != NSNotFound){
        [myHandle seekToEndOfFile];
        NSString *lostDataLeft = @"lost data on the left
leg\n\n";
        [myHandle writeData:[lostDataLeft
dataUsingEncoding:NSUTF8StringEncoding]];

    }  else if([movPattern rangeOfString:@"0000"
options:NSRegularExpressionSearch].location != NSNotFound){
        [self.still still:myHandle];
        self.stillCounter = self.stillCounter + 1;

    }  else if([movPattern rangeOfString:@"1010"
options:NSRegularExpressionSearch].location != NSNotFound){
        [self.pawing pawing:@"Right Front" myFile:myHandle];
```

```
        self.pawingRightCounter = self.pawingRightCounter + 1;

    }  else if([movPattern rangeOfString:@"0101"
options:NSRegularExpressionSearch].location != NSNotFound){
        [self.pawing pawing:@"Left Front" myFile:myHandle];
        self.pawingLeftCounter = self.pawingLeftCounter + 1;

    }  else{
        [self.moving moving:myHandle];
        self.movingCounter = self.movingCounter + 1;
    }
}  //end determineHorseBehavior


- (void)printHorseStats:(NSString *) fileName{
    NSString *pawingLeftCount = [NSString
stringWithFormat:@"\n%d",self.pawingLeftCounter];
    NSString *pawingLeftIntro = (@" seconds horse has been
pawing with left leg.\n");
    NSString *pawingLeftMessage = [pawingLeftCount
stringByAppendingString:pawingLeftIntro];
    NSString *pawingRightCount = [NSString
stringWithFormat:@"%d",self.pawingRightCounter];
    NSString *pawingRightIntro = (@" seconds horse has been
pawing with right leg.\n");
    NSString *pawingRightMessage = [pawingRightCount
stringByAppendingString:pawingRightIntro];
    NSString *stillCount = [NSString
stringWithFormat:@"%d",self.stillCounter];
    NSString *stillIntro = (@" seconds horse has been
still.\n");
    NSString *stillMessage = [stillCount
stringByAppendingString:stillIntro];
    NSString *movingCount = [NSString
stringWithFormat:@"%d",self.movingCounter];
    NSString *movingIntro = (@" seconds horse has been
moving.\n");
    NSString *movingMessage = [movingCount
stringByAppendingString:movingIntro];
    NSString *savedString = @"Horse Summary Stats\n";
    NSString *endOfLine = @"********************\n\n";

    NSFileHandle *myHandle = [NSFileHandle
fileHandleForWritingAtPath:self.documentTXTPath];
    [myHandle seekToEndOfFile];
    [myHandle writeData:[savedString
dataUsingEncoding:NSUTF8StringEncoding]];
```

```
    [myHandle seekToEndOfFile];
    [myHandle writeData:[fileName
dataUsingEncoding:NSUTF8StringEncoding]];
    [myHandle seekToEndOfFile];
    [myHandle writeData:[pawingLeftMessage
dataUsingEncoding:NSUTF8StringEncoding]];
    [myHandle seekToEndOfFile];
    [myHandle writeData:[pawingRightMessage
dataUsingEncoding:NSUTF8StringEncoding]];
    [myHandle seekToEndOfFile];
    [myHandle writeData:[stillMessage
dataUsingEncoding:NSUTF8StringEncoding]];
    [myHandle seekToEndOfFile];
    [myHandle writeData:[movingMessage
dataUsingEncoding:NSUTF8StringEncoding]];
    [myHandle seekToEndOfFile];
    [myHandle writeData:[endOfLine
dataUsingEncoding:NSUTF8StringEncoding]];
}  //end fileName

@end
```

```
//  AppDelegate.h
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;
@property (nonatomic) AVCaptureSession *session;
@property (nonatomic) AVCaptureDevice *theCamera;
@property (nonatomic) AVCaptureDeviceInput *theInputSource;
@property (nonatomic) AVCaptureStillImageOutput
*theOutputSource;
@property NSString *documentsPath;

@end
```

```
//  AppDelegate.m
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import "AppDelegate.h"

@interface AppDelegate ()

@end

@implementation AppDelegate


- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application
launch.
    return YES;
}


- (void)applicationWillResignActive:(UIApplication *)application
{
    // Sent when the application is about to move from active to
inactive state. This can occur for certain types of temporary
interruptions (such as an incoming phone call or SMS message) or
when the user quits the application and it begins the transition
to the background state.
    // Use this method to pause ongoing tasks, disable timers,
and invalidate graphics rendering callbacks. Games should use
this method to pause the game.
}


- (void)applicationDidEnterBackground:(UIApplication
*)application {
    // Use this method to release shared resources, save user
data, invalidate timers, and store enough application state
information to restore your application to its current state in
case it is terminated later.
    // If your application supports background execution, this
method is called instead of applicationWillTerminate: when the
user quits.
}
```

```
- (void)applicationWillEnterForeground:(UIApplication
*)application {
    // Called as part of the transition from the background to
the active state; here you can undo many of the changes made on
entering the background.
}


- (void)applicationDidBecomeActive:(UIApplication *)application
{
    // Restart any tasks that were paused (or not yet started)
while the application was inactive. If the application was
previously in the background, optionally refresh the user
interface.
}


- (void)applicationWillTerminate:(UIApplication *)application {
    // Called when the application is about to terminate. Save
data if appropriate. See also applicationDidEnterBackground:.
}

@end
```

```
//  ViewController.h
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import <UIKit/UIKit.h>


@interface ViewController : UIViewController

@property(nonatomic, assign) int hueLow;
@property(nonatomic, assign) int hueHigh;

@property(nonatomic, assign) int satLow;
@property(nonatomic, assign) int satHigh;

@property(nonatomic, assign) int valLow;
@property(nonatomic, assign) int valHigh;

@property(nonatomic, assign) int redHueLow;
@property(nonatomic, assign) int redHueHigh;

@property(nonatomic, assign) int redSatLow;
@property(nonatomic, assign) int redSatHigh;

@property(nonatomic, assign) int greenHueLow;
@property(nonatomic, assign) int greenHueHigh;

@property(nonatomic, assign) int greenSatLow;
@property(nonatomic, assign) int greenSatHigh;

@end
```

```
//  ViewController.m
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import "ViewController.h"
#import <opencv2/core.hpp>
#import <opencv2/imgproc.hpp>
#import <opencv2/highgui.hpp>
#import <opencv2/imgcodecs/ios.h>  //UIImageToMat
#import <iostream> //used for cout
#import <fstream> //used to write to file
#import <GameplayKit/GameplayKit.h>
#import "HorseEntity.h"
#import "AppDelegate.h"


using namespace std;
using namespace cv;

@interface ViewController (){

Mat updatedMat;
Mat drawing;
cv::Point prevLFPosition;
cv::Point currentLFPosition;
int hLFdiff;
int vLFdiff;
int movement;
int counter;
cv::Point prevRFPosition;
cv::Point currentRFPosition;
int hRFdiff;
int vRFdiff;
string prefix;
string suffix;
int frameNumber;
AppDelegate *appDelegate;
}

@property IBOutlet UIImageView *imageView;
@property IBOutlet UIImageView *imageViewOut;
@property NSTimer *timer;
@property GKRuleSystem *ruleSystem;
@property (nonatomic) BOOL didRightMove;
@property (nonatomic) BOOL didLeftMove;
@property (readwrite, nonatomic) HorseEntity *horse;
@property NSMutableString *movementPattern;
```

```
@property NSTimer *t;
@property (nonatomic) NSInteger count;
@property (nonatomic) NSInteger movementCounter;

- (IBAction)buttonTapped:(UIButton *)sender;
- (void) takePic;
- (cv::Point)determineRedBlobLocation;
- (cv::Point)determineGreenBlobLocation;
- (void) drawBlobs:(cv::Point)red :(cv::Point)green;
- (IBAction)stopButtonTapped:(id)sender;

@end //end interface declarations

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    prevLFPosition.x = 0;
    prevLFPosition.y = 0;
    currentLFPosition.x = 0;
    currentLFPosition.y = 0;
    hLFdiff = 0;
    vLFdiff = 0;
    movement = 0;

    prevRFPosition.x = 0;
    prevRFPosition.y = 0;
    currentRFPosition.x = 0;
    currentRFPosition.y = 0;
    hRFdiff = 0;
    vRFdiff = 0;
    counter = 0;

    appDelegate = (AppDelegate*)[[UIApplication
sharedApplication] delegate];

    self.movementCounter = 0;

    _ruleSystem = [[GKRuleSystem alloc] init];
    NSPredicate *rightMovementTest = [NSPredicate
predicateWithFormat:@"!((($hRFdiff.intValue > -10) && ($hRFdiff
.intValue < 10)) && (($vRFdiff.intValue > -10) &&
($vRFdiff.intValue < 10)))"];
    [_ruleSystem addRule:[GKRule
ruleWithPredicate:rightMovementTest assertingFact:@"rightMoved"
grade:1]];
```

```
    NSPredicate *leftMovementTest = [NSPredicate
predicateWithFormat:@"!((($hLFdiff.intValue > -10) && ($hLFdiff
.intValue < 10)) && (($vLFdiff.intValue > -10) &&
($vLFdiff.intValue < 10)))"];
    [_ruleSystem addRule:[GKRule
ruleWithPredicate:leftMovementTest assertingFact:@"leftMoved"
grade:1]];

    _horse = [[HorseEntity alloc] init];
    _movementPattern = [NSMutableString new];

    cout << self.greenHueLow << self.greenHueHigh <<
self.greenSatLow << self.greenSatHigh << endl;

} //end ViewDidLoad


- (void) drawBlobs:(cv::Point)red :(cv::Point)green
{
    //Note:  The drawing method will draw the point at 0,0 if no
data is received.  This is the expected behavior.
    drawing = Mat::zeros( updatedMat.size(), CV_8UC3 );
    Scalar redColor(255,0,0);
    Scalar greenColor(0,255,0);
    circle(drawing, red,  10.0 , redColor, -1);
    circle(drawing, green,  10.0 , greenColor, -1);
} //end drawBlobs


- (cv::Point)determineRedBlobLocation{
    Mat red; //holds all the red countours
    cv::Point center; //contains the center point of the bandage

    inRange(updatedMat, Scalar(self.redHueLow, self.redSatLow,
0), Scalar(self.redHueHigh, self.redSatHigh, 255), red);

    //Find largest countour - should contain the bandage
    vector<vector<cv::Point>> redContours;
    vector<Vec4i> redHierarchy;
    findContours( red, redContours, redHierarchy,
CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE );

    int largest_area=0;
    int largest_contour_index=0;
    for (int i =0;i<redContours.size(); i++)
    {
        double a=contourArea( redContours[i],false);
```

```
        if(a>largest_area)
        {
            largest_area=a;
            largest_contour_index=i;
        }
    }

    //Draw rectangle around the largest contour (should be the
bandage) and determine the center of the rectangle
    if (redContours.size() > 0)
    {
        cv::Rect box =
boundingRect(redContours[largest_contour_index]);
        center = cv::Point(((box.tl().x + box.br().x)/2),
((box.tl().y + box.br().y)/2));
        currentRFPosition.x = ((box.tl().x + box.br().x)/2);
        currentRFPosition.y = ((box.tl().y + box.br().y)/2);
    }else{
        center = (cv::Point(0), cv::Point(0));
        currentRFPosition.x = -1;
        currentRFPosition.y = -1;
    }

    return center; //return the center point
}//end determineRedBlobLocation



- (cv::Point)determineGreenBlobLocation{
    Mat green; //holds all the green countours
    cv::Point center; //contains the center point of the bandage

    inRange(updatedMat, Scalar(self.greenHueLow,
self.greenSatLow, 0), Scalar(self.greenHueHigh,
self.greenSatHigh, 255), green);

    //Find largest countour - should contain the bandage
    vector<vector<cv::Point>> greenContours;
    vector<Vec4i> greenHierarchy;
    findContours( green, greenContours, greenHierarchy,
CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE );

    int largest_area=0;
    int largest_contour_index=0;
    for (int i =0;i<greenContours.size(); i++)
    {
        double a=contourArea( greenContours[i],false);
```

```
        if(a>largest_area)
        {
            largest_area=a;
            largest_contour_index=i;
        }
    }

    //Draw rectangle around the largest contour (should be the
bandage) and determine the center of the rectangle
    if (greenContours.size() > 0)
    {
        cv::Rect box =
boundingRect(greenContours[largest_contour_index]);
        center = cv::Point(((box.tl().x + box.br().x)/2),
((box.tl().y + box.br().y)/2));
        currentLFPosition.x = ((box.tl().x + box.br().x)/2);
        currentLFPosition.y = ((box.tl().y + box.br().y)/2);
    }else{
        center = (cv::Point(0), cv::Point(0));
        currentLFPosition.x = -1;
        currentLFPosition.y = -1;
    }
    return center; //return the center point
}//end determineGreenBlobLocation




- (IBAction)buttonTapped:(UIButton *)sender //start Button
{
    self.t = [NSTimer scheduledTimerWithTimeInterval: 0.5
target: self selector:@selector(takePic) userInfo: nil
repeats:YES];
} //end buttonTapped


- (void)takePic {
    self.count = self.count + 1;

    //Saving image to the application Sandbox Documents folder
    NSString *counterStr = [@(self.count) stringValue];
    NSString *suf = @".jpg";
    NSString *fileName = [NSString stringWithFormat:@"%@%@",
counterStr, suf];
    NSString *filePath = [appDelegate.documentsPath
stringByAppendingPathComponent:fileName];
```

```
    NSString *suf2 = @"_dots.jpg";
    NSString *fileName2 = [NSString stringWithFormat:@"%@%@",
counterStr, suf2];
    NSString *filePath2 = [appDelegate.documentsPath
stringByAppendingPathComponent:fileName2];

    AVCaptureConnection *theConnection =
[appDelegate.theOutputSource
connectionWithMediaType:AVMediaTypeVideo];
    theConnection.videoOrientation =
AVCaptureVideoOrientationLandscapeRight;

    [appDelegate.theOutputSource
captureStillImageAsynchronouslyFromConnection:theConnection
completionHandler:^(CMSampleBufferRef imageDataSampleBuffer,
NSError *error) {
        NSData *imageData = [AVCaptureStillImageOutput
jpegStillImageNSDataRepresentation:imageDataSampleBuffer];
        UIImage *theImage = [UIImage imageWithData:imageData];

    if (imageData == NULL){
                NSLog(@"Error while taking photo.");
            } else{
                self.movementCounter += 1;

                //NOTE: This translation keeps the Mat in RGB
format -- it does NOT store it in BGR format like regular
OpenCV!
                UIImageToMat(theImage, updatedMat); //convert
from iOS imageView to openCV mat format

                //Must use RBG to HSV NOT the BGR like in the
non-iOS version!
                 cvtColor(updatedMat, updatedMat,
COLOR_RGB2HSV); //convert to a HSV colorspace

                //Blur the image to remove noise
                medianBlur(updatedMat, updatedMat, 9);

                cv::Point redBlobLocation = [self
determineRedBlobLocation];
                cv::Point greenBlobLocation = [self
determineGreenBlobLocation];
                [self drawBlobs:redBlobLocation
:greenBlobLocation];
```

```
                [imageData writeToFile:filePath atomically:YES];
//Write the file to the application sandbox
                NSData *convertedDrawing =
UIImagePNGRepresentation(MatToUIImage(drawing));
                [convertedDrawing writeToFile:filePath2
atomically:YES];

                self.imageView.image = theImage;
                self.imageViewOut.image = MatToUIImage(drawing);
//display the bandage locations

                if (((currentRFPosition.x == -1) &&
(currentRFPosition.y == -1)) && ((currentLFPosition.x == -1) &&
(currentLFPosition.y == -1))){
                    NSString *bothLostData = @"99";
                    [self.movementPattern
appendString:bothLostData];
                } else if ((currentRFPosition.x == -1) &&
(currentRFPosition.y == -1)){
                    NSString *rightLostData = @"88";
                    [self.movementPattern
appendString:rightLostData];
                } else if ((currentLFPosition.x == -1) &&
(currentLFPosition.y == -1)){
                    NSString *leftLostData = @"77";
                    [self.movementPattern
appendString:leftLostData];
                } else {
                    hLFdiff = (currentLFPosition.x -
prevLFPosition.x);
                    vLFdiff = (currentLFPosition.y -
prevLFPosition.y);
                    prevLFPosition = currentLFPosition;

                    hRFdiff = (currentRFPosition.x -
prevRFPosition.x);
                    vRFdiff = (currentRFPosition.y -
prevRFPosition.y);
                    prevRFPosition = currentRFPosition;

                    self.ruleSystem.state[@"hRFdiff"] =
@(hRFdiff);
                    self.ruleSystem.state[@"vRFdiff"] =
@(vRFdiff);
                    self.ruleSystem.state[@"hLFdiff"] =
@(hLFdiff);
```

```
                    self.ruleSystem.state[@"vLFdiff"] =
@(vLFdiff);

                    [self.ruleSystem reset];
                    [self.ruleSystem evaluate];

                    self.didRightMove = ([self.ruleSystem
gradeForFact:@"rightMoved"] > 0.0);
                    self.didLeftMove = ([self.ruleSystem
gradeForFact:@"leftMoved"] > 0.0);
                    NSString *leftBool2String =
(self.didLeftMove) ? @"1" : @"0";
                    NSString *rightBool2String =
(self.didRightMove) ? @"1" : @"0";
                    [self.movementPattern
appendString:rightBool2String];
                    [self.movementPattern
appendString:leftBool2String];
                }
            }

        if (self.movementCounter == 2){
            [self.horse
determineHorseBehavior:self.movementPattern];
            [self.horse printHorseStats:fileName];
            [self.movementPattern setString:@""]; //This resets
the movementPattern
            self.movementCounter = 0;
        }
    }];  //end asynchronous call
} //end takePic



- (IBAction)stopButtonTapped:(id)sender {
    [self.t invalidate];
    self.t = nil;
    [appDelegate.theCamera unlockForConfiguration];
} //end stopButtonTapped


- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
} //end didReceiveMemoryWarning
@end
```

```
//  InitialViewController.h
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import <UIKit/UIKit.h>

@interface InitialViewController: UIViewController

//Hue
@property (weak, nonatomic) IBOutlet UISlider *hueLowSlider;
@property (weak, nonatomic) IBOutlet UILabel *hueLowLabel;
@property (weak, nonatomic) IBOutlet UISlider *hueHighSlider;
@property (weak, nonatomic) IBOutlet UILabel *hueHighLabel;

//Saturation
@property (weak, nonatomic) IBOutlet UISlider *satLowSlider;
@property (weak, nonatomic) IBOutlet UILabel *satLowLabel;
@property (weak, nonatomic) IBOutlet UISlider *satHighSlider;
@property (weak, nonatomic) IBOutlet UILabel *satHighLabel;

@property (readwrite, nonatomic)NSArray *paths;
@property (readwrite, nonatomic)NSString *documentsDirectory;
@property (readwrite, nonatomic)NSString *documentTXTPath;

@end
```

```
//  InitialViewController.m
//  Final Program for Dissertation
//  Created by Megan M. Burton

#import "InitialViewController.h"
#import "ViewController.h" //Must add this to access the correct
variables in the segue
#import "AppDelegate.h"
#import <opencv2/core.hpp>
#import <opencv2/imgproc.hpp>
#import <opencv2/highgui.hpp>
#import <opencv2/imgcodecs/ios.h>  //UIImageToMat
#import <iostream> //used for cout

using namespace std;
using namespace cv;

@interface InitialViewController (){

Mat imgHSV;
Mat imgThresholded;
vector<Mat> channels;
int hueLow;
int hueHigh;
int satLow;
int satHigh;
int redHueLow;
int redHueHigh;
int redSatLow;
int redSatHigh;
int greenHueLow;
int greenHueHigh;
int greenSatLow;
int greenSatHigh;
AppDelegate *appDelegate;
}
@property (weak, nonatomic) IBOutlet UIImageView *ImageView;
@property (weak, nonatomic) IBOutlet UIImageView *initialImage;
@property (weak, nonatomic) IBOutlet UIImageView
*thresholdImage;
@property (weak, nonatomic) IBOutlet UIButton *usePicButton;
@property (weak, nonatomic) IBOutlet UIButton *takePicButton;
@property (weak, nonatomic) IBOutlet UIButton *redButton;
@property (weak, nonatomic) IBOutlet UIButton *greenButton;
@property (weak, nonatomic) IBOutlet UIButton *startButton;
@property NSString* filePath;
```

```
@end //end interface declarations


@implementation InitialViewController

- (IBAction)takePicButtonPressed:(id)sender {
    AVCaptureConnection *theConnection =
[appDelegate.theOutputSource
connectionWithMediaType:AVMediaTypeVideo];
    theConnection.videoOrientation =
AVCaptureVideoOrientationLandscapeRight;

    NSString *fileName = @"0.jpg";
    NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    appDelegate.documentsPath = [paths objectAtIndex:0]; //Get
the documents directory
    self.filePath = [appDelegate.documentsPath
stringByAppendingPathComponent:fileName]; //Add the file name

    [appDelegate.theOutputSource
captureStillImageAsynchronouslyFromConnection:theConnection
completionHandler:^(CMSampleBufferRef imageDataSampleBuffer,
NSError *error) {
        NSData *imageData = [AVCaptureStillImageOutput
jpegStillImageNSDataRepresentation:imageDataSampleBuffer];
        UIImage *theImage = [UIImage imageWithData:imageData];
        [imageData writeToFile:self.filePath atomically:YES];
//Write the file to application sandbox
        self.initialImage.image = theImage;
        UIImageWriteToSavedPhotosAlbum(theImage, self, nil,
nil);  //Write the file to the camera roll
    }];//end asynchronous call

    self.usePicButton.enabled = YES;

}//end takePicButtonPressed


- (IBAction)usePic:(id)sender {
    self.ImageView.hidden = YES; //Needed so the live camera
preview is no longer shown
    self.takePicButton.enabled = NO;
    UIImage *initialImage = [UIImage imageNamed:self.filePath];
    UIImageToMat(initialImage, imgHSV);
    cvtColor(imgHSV, imgHSV, COLOR_RGB2HSV);
```

```
    inRange(imgHSV, Scalar(hueLow, satLow, 0), Scalar(hueHigh,
satHigh, 255), imgThresholded); //Threshold the image

    //morphological opening (remove small objects from the
foreground)
    erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(3, 3)) );
    dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(3, 3)) );

    //morphological closing (fill small holes in the foreground)
    dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
    erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );

    self.thresholdImage.image = MatToUIImage(imgThresholded);
    self.thresholdImage.hidden = NO;
    self.usePicButton.enabled = NO;
    self.redButton.enabled = YES;
    self.satLowSlider.enabled = YES;
    self.satHighSlider.enabled = YES;
    self.hueLowSlider.enabled = YES;
    self.hueHighSlider.enabled = YES;
}// end usePic


- (IBAction)redButtonPressed:(id)sender {
    redHueLow = (int)self.hueLowSlider.value;
    redHueHigh = (int)self.hueHighSlider.value;
    redSatLow = (int)self.satLowSlider.value;
    redSatHigh = (int)self.satHighSlider.value;
    self.redButton.enabled = NO;
    self.greenButton.enabled = YES;
}// end redButtonPressed


- (IBAction)greenButtonPressed:(id)sender {
    greenHueLow = (int)self.hueLowSlider.value;
    greenHueHigh = (int)self.hueHighSlider.value;
    greenSatLow = (int)self.satLowSlider.value;
    greenSatHigh = (int)self.satHighSlider.value;
    self.greenButton.enabled = NO;
    self.startButton.enabled = YES;
}// end greenButtonPressed
```

```
- (void)viewDidLoad {
    [super viewDidLoad];
    hueLow = 0;
    hueHigh = 179;
    satLow = 150;
    satHigh = 255;

    self.takePicButton.enabled = YES;
    self.usePicButton.enabled = NO;
    self.redButton.enabled = NO;
    self.greenButton.enabled = NO;
    self.startButton.enabled = NO;
    self.satLowSlider.enabled = NO;
    self.satHighSlider.enabled = NO;
    self.hueLowSlider.enabled = NO;
    self.hueHighSlider.enabled = NO;
    self.thresholdImage.hidden = YES;

    appDelegate = (AppDelegate*)[[UIApplication
sharedApplication] delegate];

    NSArray *allCameras = [AVCaptureDevice
devicesWithMediaType:AVMediaTypeVideo];
    for (AVCaptureDevice *camera in allCameras) {
        if ([camera position] == AVCaptureDevicePositionBack){
            appDelegate.theCamera = camera;
            break;
        }
    }

    appDelegate.session = [[AVCaptureSession alloc]init];
    appDelegate.theInputSource = [AVCaptureDeviceInput
deviceInputWithDevice:appDelegate.theCamera error:nil];

    if ([appDelegate.session
canAddInput:appDelegate.theInputSource]) {
        [appDelegate.session
addInput:appDelegate.theInputSource];
    }

    appDelegate.session.sessionPreset =
AVCaptureSessionPreset640x480;

    NSError *error = nil;
    if ( [appDelegate.theCamera lockForConfiguration:&error] ) {
        appDelegate.theCamera.flashMode = AVCaptureFlashModeOff;
```

```
        [appDelegate.theCamera
setWhiteBalanceModeLockedWithDeviceWhiteBalanceGains:AVCaptureWh
iteBalanceGainsCurrent completionHandler:nil];
    }

    appDelegate.theOutputSource = [[AVCaptureStillImageOutput
alloc]init];
    [appDelegate.session addOutput:appDelegate.theOutputSource];

    AVCaptureVideoPreviewLayer *previewLayer =
[AVCaptureVideoPreviewLayer
layerWithSession:appDelegate.session];
    previewLayer.connection.videoOrientation =
AVCaptureVideoOrientationLandscapeRight;
    UIView *aView = self.ImageView;
    previewLayer.frame = aView.bounds;
    [aView.layer addSublayer:previewLayer];
    [appDelegate.session startRunning];
}// end viewDidLoad

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}//end didReceiveMemoryWarning

//HUE SLIDERS
- (IBAction)hueLowSliderValueChanged:(id)sender {
    hueLow = (int)self.hueLowSlider.value;
    self.hueLowLabel.text = [NSString stringWithFormat:@"%i",
(int)self.hueLowSlider.value];

    inRange(imgHSV, Scalar(hueLow, satLow, 0), Scalar(hueHigh,
satHigh, 255), imgThresholded); //Threshold the image

    //morphological opening (remove small objects from the
foreground)
    erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(3, 3)) );
    dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(3, 3)) );

    //morphological closing (fill small holes in the foreground)
    dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
    erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
    self.thresholdImage.image = MatToUIImage(imgThresholded);
```

```
}// end hueLowSliderValueChanged

- (IBAction)hueHighSliderValueChanged:(id)sender {
    hueHigh = (int)self.hueHighSlider.value;
    self.hueHighLabel.text = [NSString stringWithFormat:@"%i",
(int)self.hueHighSlider.value];
    inRange(imgHSV, Scalar(hueLow, satLow, 0), Scalar(hueHigh,
satHigh, 255), imgThresholded); //Threshold the image

    //morphological opening (remove small objects from the
foreground)
    erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(3, 3)) );
    dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(3, 3)) );

    //morphological closing (fill small holes in the foreground)
    dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
    erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
    self.thresholdImage.image = MatToUIImage(imgThresholded);
}//end hueHighSliderValueChanged


//SATURATION SLIDERS
- (IBAction)satLowSliderValueChanged:(id)sender {
    satLow = (int)self.satLowSlider.value;
    self.satLowLabel.text = [NSString stringWithFormat:@"%i",
(int)self.satLowSlider.value];
    inRange(imgHSV, Scalar(hueLow, satLow, 0), Scalar(hueHigh,
satHigh, 255), imgThresholded); //Threshold the image

    //morphological opening (remove small objects from the
foreground)
    erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(3, 3)) );
    dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(3, 3)) );

    //morphological closing (fill small holes in the foreground)
    dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
    erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
    self.thresholdImage.image = MatToUIImage(imgThresholded);
}// satLowSliderValueChanged
```

```
- (IBAction)satHighSliderValueChanged:(id)sender {
    satHigh = (int)self.satHighSlider.value;
    self.satHighLabel.text = [NSString stringWithFormat:@"%i",
(int)self.satHighSlider.value];
    inRange(imgHSV, Scalar(hueLow, satLow, 0), Scalar(hueHigh,
satHigh, 255), imgThresholded); //Threshold the image

    //morphological opening (remove small objects from the
foreground)
    erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(3, 3)) );
    dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(3, 3)) );

    //morphological closing (fill small holes in the foreground)
    dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
    erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );

    self.thresholdImage.image = MatToUIImage(imgThresholded);
}//end satHighSliderValueChanged


- (void)prepareForSegue:(UIStoryboardSegue *)segue
sender:(id)sender {
    self.paths =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,NSUserDo
mainMask, YES);
    self.documentsDirectory = [self.paths objectAtIndex:0];
    self.documentTXTPath = [self.documentsDirectory
stringByAppendingPathComponent:@"SliderValues.txt"];

    NSFileManager *fileManager = [NSFileManager defaultManager];
    if(![fileManager fileExistsAtPath:self.documentTXTPath])
    {
        NSString *savedString = @"Slider
Values\n*****START*****\n";
        [savedString writeToFile:self.documentTXTPath
atomically:YES];
    } else{
        NSFileHandle *myHandle = [NSFileHandle
fileHandleForWritingAtPath:self.documentTXTPath];
        [myHandle seekToEndOfFile];
        NSString *start = @"*****START*****\n";
```

```
        [myHandle writeData:[start
dataUsingEncoding:NSUTF8StringEncoding]];
    }

    ViewController *vc = [segue destinationViewController];

    vc.redHueLow = redHueLow;
    vc.redHueHigh = redHueHigh;
    vc.greenHueLow = greenHueLow;
    vc.greenHueHigh = greenHueHigh;
    vc.redSatLow = redSatLow;
    vc.redSatHigh = redSatHigh;
    vc.greenSatLow = greenSatLow;
    vc.greenSatHigh = greenSatHigh;

    //The following section writes the slider values to the
SliderValues.txt file
    NSFileHandle *myHandle = [NSFileHandle
fileHandleForWritingAtPath:self.documentTXTPath];
    [myHandle seekToEndOfFile];
    NSDate *now = [NSDate date];
    NSString *time = [NSString stringWithFormat:@"%@\n",now];
    [myHandle writeData:[time
dataUsingEncoding:NSUTF8StringEncoding]];

    //RED
    [myHandle seekToEndOfFile];
    NSString *savedString = [NSString stringWithFormat:@"Red Hue
Low: %i\n",redHueLow];
    [myHandle writeData:[savedString
dataUsingEncoding:NSUTF8StringEncoding]];

    [myHandle seekToEndOfFile];
    savedString = [NSString stringWithFormat:@"Red Hue High:
%i\n",redHueHigh];
    [myHandle writeData:[savedString
dataUsingEncoding:NSUTF8StringEncoding]];

    [myHandle seekToEndOfFile];
    savedString = [NSString stringWithFormat:@"Red Sat Low:
%i\n",redSatLow];
    [myHandle writeData:[savedString
dataUsingEncoding:NSUTF8StringEncoding]];

    [myHandle seekToEndOfFile];
    savedString = [NSString stringWithFormat:@"Red Sat High:
%i\n",redSatHigh];
```

```
    [myHandle writeData:[savedString
dataUsingEncoding:NSUTF8StringEncoding]];

    //GREEN
    [myHandle seekToEndOfFile];
    savedString = [NSString stringWithFormat:@"Green Hue Low:
%i\n",greenHueLow];
    [myHandle writeData:[savedString
dataUsingEncoding:NSUTF8StringEncoding]];

    [myHandle seekToEndOfFile];
    savedString = [NSString stringWithFormat:@"Green Hue High:
%i\n",greenHueHigh];
    [myHandle writeData:[savedString
dataUsingEncoding:NSUTF8StringEncoding]];

    [myHandle seekToEndOfFile];
    savedString = [NSString stringWithFormat:@"Green Sat Low:
%i\n",greenSatLow];
    [myHandle writeData:[savedString
dataUsingEncoding:NSUTF8StringEncoding]];

    [myHandle seekToEndOfFile];
    savedString = [NSString stringWithFormat:@"Green Sat High:
%i\n\n",greenSatHigh];
    [myHandle writeData:[savedString
dataUsingEncoding:NSUTF8StringEncoding]];
}// end prepareForSegue
@end
```

APPENDIX B: SOFTWARE SCREEN LAYOUT

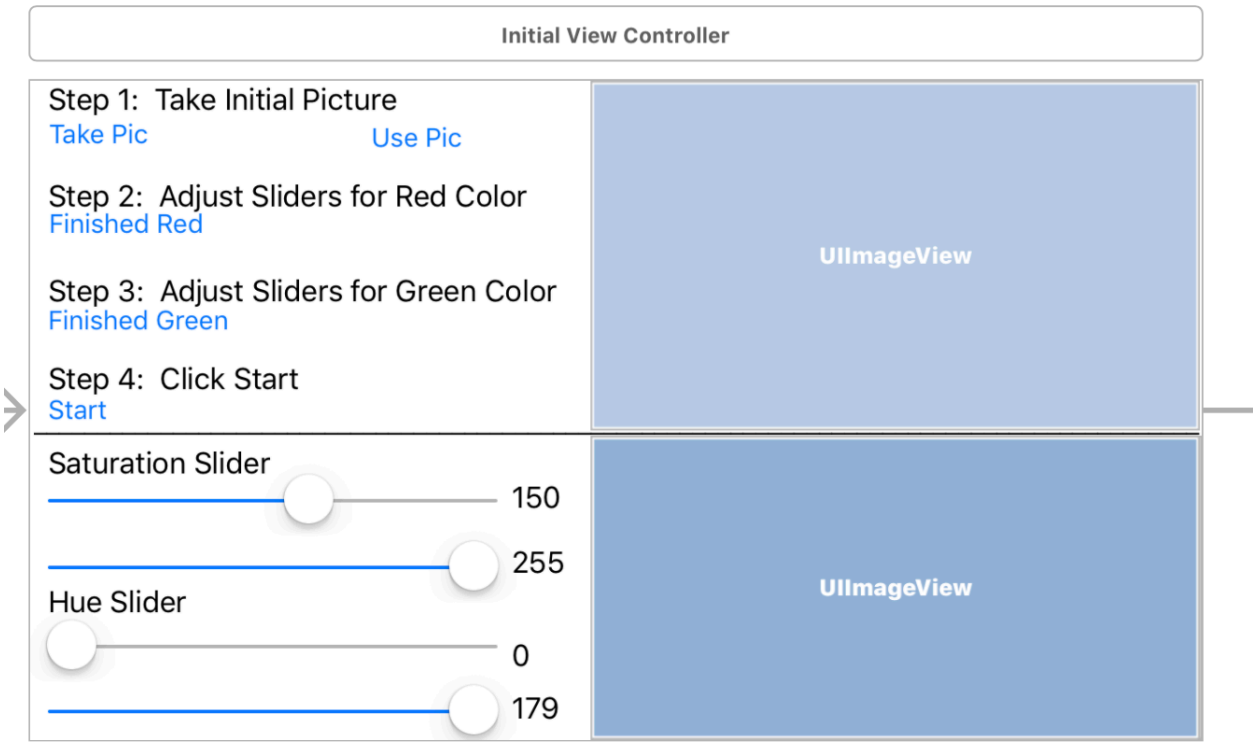

Figure B.1: Initial View Controller Screen Layout

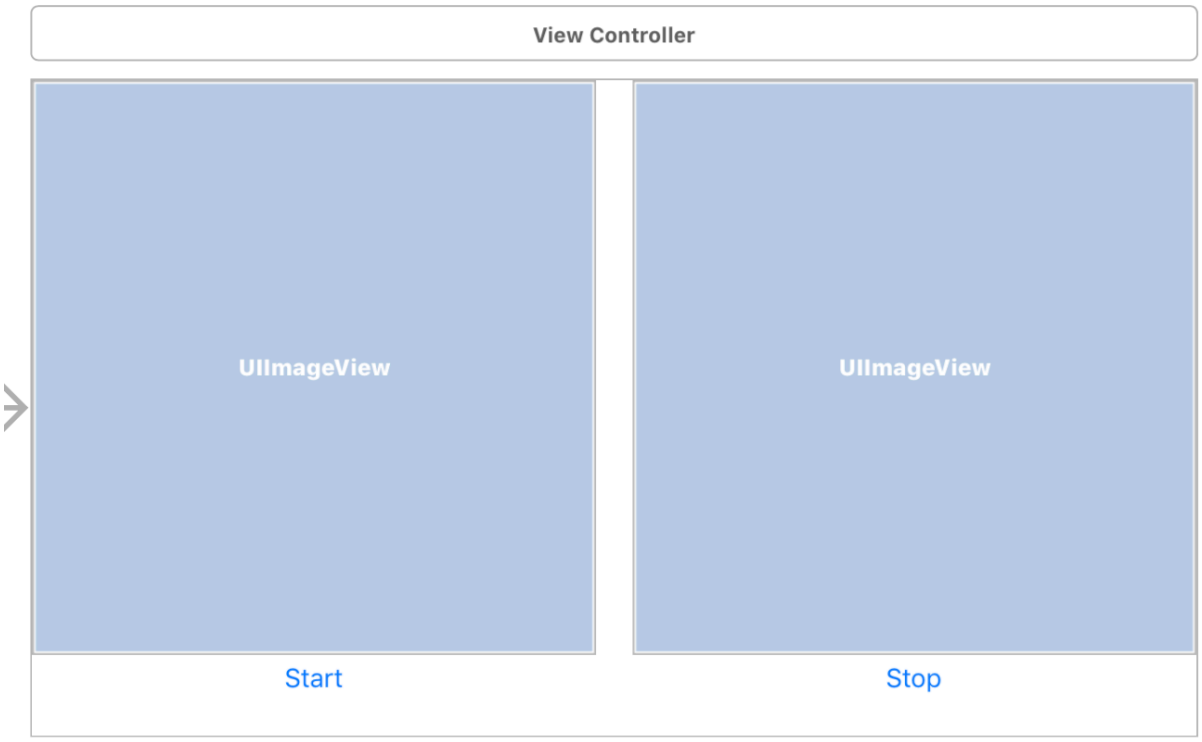

Figure B.2: Main View Controller Screen Layout

APPENDIX C: INSTITUTIONAL REVIEW BOARD APPROVAL

APPROVED

2016-2851

ANIMAL SUBJECTS REVIEW FORM

PRINCIPAL INVESTIGATOR: Dr. David Umphress
RANK/TITLE: Professor
DEPARTMENT: Computer Science & Software Engineering
COLLEGE/SCHOOL: Samuel Ginn College of Engineering
CAMPUS ADDRESS: 3127E Shelby Center
CAMPUS PHONE #: 334-844-6335
E-MAIL: umphrda@auburn.edu
FAX #: NA

[X] Check if PI will serve as faculty advisor to the Lead Graduate Student or Resident associated with this activity.

LEAD GRADUATE STUDENT/RESIDENT: Megan M. Burton
RANK/TITLE: PhD Student – Distance Learning (not on campus)
DEPARTMENT: Computer Science & Software Engineering
CAMPUS PHONE #: 256-426-5435 (cell)
EMAIL: burtomm@auburn.edu
FAX #: NA

CO-INVESTIGATOR: NA
RANK/TITLE:
DEPARTMENT:
CAMPUS PHONE #:
EMAIL:
FAX #:

[ ] Check box if this protocol has more than one co-investigator. Additional co-investigators should be listed on page 2.

PROJECT TITLE: Use of Smartphone To Determine Equine Behavior
STARTING DATE: ~~1 Feb 2016~~ 3/17/2016 *(Must not be prior to IACUC approval)*
EXPIRATION DATE: ~~1 Feb 2019~~ 3/16/2019 *(Must not exceed three years)*

Is any part of the funding from a U.S. PUBLIC HEALTH SERVICE AGENCY: YES [ ] NO [X]

REQUIRED SIGNATURES

The information contained on this form provides an accurate description of the animal care and use protocol which will be followed. I agree to abide by governmental regulations and university policies concerning the use of animals. I will allow veterinary oversight to be provided to animals showing evidence of pain or illness. If the information provided for this project concerning animal use should be revised, or procedures changed, I will so notify the committee of those changes in writing, and no proposed changes will be implemented until full IACUC approval has been granted.

X _____ 15 Dec 2015
Principal Investigator — Date

Medical care for animals will be available and provided as indicated by a qualified veterinarian. By accepting this responsibility, the veterinarian is providing assurance that any personal interest he/she might have in the project will not conflict with his/her responsibility for the provision of adequate veterinary care for the animals. Furthermore, the veterinarian provides assurance of review and consultation on the proper use of anesthetics and pain relieving medications for any painful procedures.

Mark Lowe DVM
Project Veterinarian Name (print or type)

X _____ 12-14-15
Project Veterinarian Signature — Date

NA
Unit Veterinarian Name (print or type)

X NA
Unit Veterinarian Signature — Date

Dean Hendrix (for Kai Chang)
Departmental Chairperson Name (print or type)

X _____ 12/17/2015
Departmental Chairperson Signature — Date

X _____ 14 Dec 2015
Lead Graduate Student/Resident signature — Date

X _____ 3/17/2016
*IACUC Chair Signature — Date
*IACUC Chair signs the protocol after IACUC approval has been granted

ASRF 04/2012
Page 1