

Automatic Dynamic Tracking of Horse Head Facial Features in Video Using Image Processing Techniques

Jason E. Doyle

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Mechanical Engineering

Mary E. Kasarda, Chair
Amos L. Abbott
Virginia A. Buechner-Maxwell
Hampton C. Gabler III
Kevin B. Kochersberger

December 12, 2018
Blacksburg, Virginia

Keywords: Equine Pain Face, Horse Grimace Scale, Computer Vision, Facial Feature
Detection

Copyright 2018, Jason E. Doyle

Automatic Dynamic Tracking of Horse Head Facial Features in Video Using Image Processing Techniques

Jason E. Doyle

(ABSTRACT)

The well-being of horses is very important to their care takers, trainers, veterinarians, and owners. This thesis describes the development of a non-invasive image processing technique that allows for automatic detection and tracking of horse head and ear motion, respectively, in videos or camera feed, both of which may provide indications of horse pain, stress, or well-being. The algorithm developed here can automatically detect and track head motion and ear motion, respectively, in videos of a standing horse. Results demonstrating the technique for nine different horses are presented, where the data from the algorithm is utilized to plot absolute motion vs. time, velocity vs. time, and acceleration vs. time for the head and ear motion, respectively, of a variety of horses and ponies. Two-dimensional plotting of x and y motion over time is also presented. Additionally, results of pilot work in eye detection in light colored horses is also presented. Detection of pain in horses is particularly difficult because they are prey animals and have mechanisms to disguise their pain, and these instincts may be particularly strong in the presence of an unknown human, such as a veterinarian. Current state-of-the art for detecting pain in horses primarily involves invasive methods, such as heart rate monitors around the body, drawing blood for cortisol levels, and pressing on painful areas to elicit a response, although some work has been done for humans to sort and score photographs subjectively in terms of a “horse grimace scale.” The algorithms developed in this thesis are the first that the author is aware of for exploiting proven image processing approaches from other applications for development of an automatic tool for detection and tracking of horse facial indicators. The algorithms were done in common open source programs Python and OpenCV, and standard image processing approaches including Canny Edge detection Hue, Saturation, Value color filtering, and contour tracking were utilized in algorithm development. The work in this thesis provides the foundational development of a non-invasive and automatic detection and tracking program for horse head and ear motion, including demonstration of the viability of this approach using videos of standing horses. This approach lays the groundwork for robust tool development for monitoring horses using noninvasive methods and without the required presence of humans in such applications as post-operative monitoring, foaling, evaluation of performance horses in competition and/or training, as well as for providing data for research on animal welfare, among other scenarios.

Automatic Dynamic Tracking of Horse Head Facial Features in Video Using Image Processing Techniques

Jason E. Doyle

(GENERAL AUDIENCE ABSTRACT)

There are many things that cause pain in horses, including improper saddle fit, inadequate care, laminitis, lameness, surgery, and colic, among others. The well-being of horses is very important to their care takers, trainers, veterinarians, and owners. Monitoring the well-being of horses is particularly important in many scenarios including post-operative monitoring, therapeutic riding programs, racing, dressage, and rodeo events, among numerous other activities. This thesis describes the development of a computer-based image processing technique for automatic detection and tracking of both horse head and ear motion, respectively, in videos of standing horses. The techniques developed here allow for the collection of data on head and ear motion over time, facilitating analysis of these motions that may provide reliable indicators of horse pain, stress, or well-being. Knowing if a horse is in pain is difficult because horses are prey animals that have mechanisms in place that minimize the display of pain so that they do not become easy targets for predators. Computer vision systems, like the one developed here, may be well suited to detect subtle changes in horse behavior for detecting distress in horses. The ability to remotely and automatically monitor horse well-being by exploiting computer-based image-processing techniques will create significant opportunities to improve the welfare of horses. The work presented here looks at the first use of image-processing approaches to detect and track facial features of standing horses in videos to help facilitate the development of automatic pain and stress detection in videos and camera feeds for owners, veterinarians, and horse-related organizations, among others.

Contents

List of Figures	viii
List of Tables	xv
1 Introduction & Review of Literature	1
1.1 Noninvasive Detection of Horse Pain	2
1.2 Recognition of Human Facial Features in Photos and Videos	3
1.3 Applications of Image Processing on Animals	4
2 Image Processing Background	6
2.1 Python Programming Language	6
2.2 The OpenCV Library	7
2.3 Image and Video in Computer Vision	7
2.4 Linear Filtering	8
2.5 Hue, Saturation, Value (HSV) Colorspace	10
2.6 Canny Edge Detection	11
2.7 Contours	12
3 Pilot Study for Head Tracking	14
3.1 Grab Cuts	14
3.2 Dots on The Horse	14
3.3 ORB (Oriented FAST and Rotated BRIEF)	15
3.3.1 Attempt 1 of 3 for Halter Tracking Through ORB Feature Matching	15
3.3.2 Attempt 2 of 3 for Halter Tracking Through ORB Feature Matching	16
3.3.3 Attempt 3 of 3 for Halter Tracking Through ORB Feature Matching	17
3.4 Halter Tracking using Hough Lines	17

4	Methodology	19
4.1	Experimental Testing Procedure	19
4.1.1	File Naming Conventions	20
4.2	Overview of Detection and Tracking Code	21
4.3	Head Tracking through halter identification	23
4.4	Ear Tracking	28
4.5	Pilot Work on Eye Detection	30
4.6	Data Presentation	31
4.6.1	Motion Tracking Graphs	32
4.6.2	Velocity and Acceleration Tracking	33
4.7	Accuracies	34
5	Results	37
5.1	Data Presentation	37
5.1.1	Head tracking	37
5.1.2	Ear Tracking	44
5.1.3	Filtered Data	48
5.2	Head Tracking and Ear Tracking Results	52
5.2.1	Big Red	52
5.2.2	White Star	55
5.2.3	Big Benny	58
5.2.4	Chief	62
5.2.5	Mosfet	65
5.2.6	Shaggy Pony	68
5.2.7	The Dude	71
5.2.8	Titanic	74
5.2.9	Tethys	77
5.3	Summary	80

6 Conclusion	82
6.1 Contributions	82
6.1.1 Recommendations	83
6.2 Future Work	84
6.2.1 Areas of Improvement	84
Bibliography	85
Appendices	88
Appendix A Release Form	89
Appendix B Video Data	92
B.1 mvi_4891.mov	92
B.2 mvi_4892.mov	92
B.3 mvi_4893.mov	92
B.3.1 edit 00	92
B.3.2 edit 01	95
B.4 mvi_4894.mov	97
B.4.1 edit 01	97
B.4.2 edit 02	97
B.4.3 edit 03	98
B.5 mvi_4895.mov	102
B.5.1 edit 00	102
B.6 mvi_4896.mov	104
B.7 mvi_4897.mov	104
B.8 mvi_4898.mov	104
B.9 mvi_4899.mov	107
B.10 mvi_4900.mov	109
B.11 mvi_4901.mov	111

B.12	<code>mvi_4902.mov</code>	113
B.13	<code>mvi_4903.mov</code>	115
B.14	<code>mvi_4904.mov</code>	117
B.15	<code>mvi_4905.mov</code>	117
B.16	<code>mvi_4906.mov</code>	117
B.17	<code>mvi_4907.mov</code>	117
B.18	<code>mvi_4908.mov</code>	118
B.19	<code>mvi_4909.mov</code>	118
B.20	<code>mvi_4910.mov</code>	118
B.21	<code>mvi_4911.mov</code>	118
B.22	<code>mvi_4912.mov</code>	119
B.23	<code>mvi_4913.mov</code>	119
B.24	<code>mvi_4914.mov</code>	119
B.25	<code>mvi_4915.mov</code>	121
B.26	<code>mvi_4916.mov</code>	123
B.27	<code>mvi_4917.mov</code>	123
B.28	<code>mvi_4918.mov</code>	125
B.29	<code>mvi_4919.mov</code>	125
Appendix C Equipment and Code		126
C.1	Equipment	126
C.2	Software	126
C.3	OpenCV Installation	126
C.4	Source Code	127
C.4.1	<code>horse_head_tacker.py</code>	127
C.4.2	<code>csv_to_graph.py</code>	146

List of Figures

1.1	2D example of a plane created to separate two classes. A non-linear plane shown on the left and a linear plane on the right.	4
2.1	How humans see an image and the matrix, in yellow, of how computers and cameras see the image.	7
2.2	The left image show an OpenCV image displayed with Matplotlib without switching to RGB order. The right image is the original image.	8
2.3	Example of a kernel with the X-direction Sobel filter on the left and Y-direction Sobel filter on the right.	9
2.4	Simplified example of an image.	9
2.5	The overhang of a kernel when it is near the borders of an image.	10
2.6	Visual representation of the different borders in OpenCV.	10
2.7	Visual representation of HSV, where each thin slice represents a hue	11
2.8	Add caption.	11
2.9	A visual representation of the minVal and maxVal. 'A' is an edge because it is above maxVal, 'B' is not an edge because it is below the maxVal and not connected to an edge above, but 'C' is an edge because it is connected to 'A' which is above maxVal.	12
3.1	ORB halter point matching attempt one. Lines should connect the same point on each halter.	16
3.2	ORB halter point matching attempt two. Lines should connect the same point on each halter.	16
3.3	ORB halter point matching attempt three. This shows the best results where the lines connect to the same point on each halter.	17
4.1	Images of the horses recorded for this thesis and their research names (not real names).	20
4.2	Flow chart of program	22

4.3	A graphical representation of RGB coordinates given values for HSV. The top color bar shows the separation of red. 0 to 360 degrees on the color wheel is represented by the values 0 to 255 in OpenCV.	24
4.4	A single frame of Big Red. On the left shows the original color image with the bounding boxes drawn, and the right image shows the halter after HSV filtering which is used to create bounding boxes.	25
4.5	The parts of the halter.	26
4.6	Flow chart of the function halter_filter	27
4.7	The original image of Big Red a light colored horse (left) and the HSV filter image (right).	29
4.8	The original image of White Star a dark colored horse (left) and the HSV filter of White Star head (right). Shadows and the background made filtering the dark horses more difficult as can be seen in the left image as part of the head is missing and the background shows up in white.	29
4.9	The blue arrows show how the program starts at the top halter point in green, goes straight up then follows the red line to the ear tip.	30
4.10	Eye closed changes the detection size, shape, and number of detections.	31
4.11	Right motion of the ear translates to downward motion of the maroon line on the ear graphs.	32
4.12	Just like the ear right motion of the head translates to downward motion of the maroon line on the halter graphs.	32
4.13	Down motion of the ear translates to a downward motion of the orange line on the ear graphs.	33
4.14	Just like the ear down motion of the head translates to downward motion of the maroon line on the halter graphs.	33
4.15	Full image of White Star(a), and a zoomed in image of the ear(b) to show the eight pixel distance between authors ear tip location(green) and the computers ear tip location(blue). The diameter of the green dot is 5 pixels and ten pixels for the blue.	36
5.1	Photo of The Dude	39
5.2	Head motion graph of The Dude.	39
5.3	2-Dimensional Head motion graph of The Dude.	40
5.4	The first frame of the video to show the initial head position.	41

5.5	Frame 61 where Big Red looks past the camera and the head tracker tracks the opposite side of the halter that is denoted by the left most green box with a blue dot at the center that might be difficult to see.	42
5.6	A frame shortly after Big Red looks forward again after looking at camera. Frame 121 denoted in the top graph of Figure 5.7.	42
5.7	The first ten seconds of clip 4893 edit 00. The graph shows when Big Red looks to the left past the camera. A, B, C correspond to Figures 5.4, 5.5, 5.6.	43
5.8	Photo of Shaggy Pony	45
5.9	Ear motion of Shaggy Pony	46
5.10	2D graph of head and ear motion of Shaggy Pony	46
5.11	Ear motion of White Star	47
5.12	Photo of White Star with ear forward and ear back.	47
5.13	Filtered head motion of Big Red.	48
5.14	Filtered ear motion of Big Red.	49
5.15	Filtered head motion of Shaggy Pony.	50
5.16	Filtered ear motion of Shaggy Pony.	51
5.17	Image of Big Red.	53
5.18	The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).	53
5.19	Orange is vertical motion, and maroon is horizontal motion of the ear tip.	54
5.20	2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicted by purple to green to yellow, and head motion is indicated by purple to red to yellow.	54
5.21	Image of White Star.	55
5.22	The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).	56
5.23	Orange is vertical motion, and maroon is horizontal motion of the ear tip.	56
5.24	2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicted by purple to green to yellow, and head motion is indicated by purple to red to yellow.	57
5.25	Image of Big Benny.	58

5.26	The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).	59
5.27	Orange is vertical motion, and maroon is horizontal motion of the ear tip.	59
5.28	2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicted by purple to green to yellow, and head motion is indicated by purple to red to yellow.	60
5.29	Image of Big Benny to show the head tracker tracking the chinpiece shown as the blue dot. Also notice the green boxes on the back that show the program thinks the blanket is part of the halter.	60
5.30	Another image from the same clip that show the head tracker now tracking the cheekpiece. The tracking o different parts of the halter show up as near vertical lines on position, velocity, acceleration graphs.	61
5.31	Image of Chief.	62
5.32	The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).	63
5.33	Orange is vertical motion, and maroon is horizontal motion of the ear tip.	63
5.34	2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicted by purple to green to yellow, and head motion is indicated by purple to red to yellow.	64
5.35	Image of Mosfet.	65
5.36	The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).	66
5.37	Orange is vertical motion, and maroon is horizontal motion of the ear tip.	66
5.38	2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicted by purple to green to yellow, and head motion is indicated by purple to red to yellow.	67
5.39	Image of Shaggy Pony.	68
5.40	The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).	69
5.41	Orange is vertical motion, and maroon is horizontal motion of the ear tip.	69
5.42	2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicted by purple to green to yellow, and head motion is indicated by purple to red to yellow.	70
5.43	Image of The Dude.	71

5.44	The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).	72
5.45	Orange is vertical motion, and maroon is horizontal motion of the ear tip.	72
5.46	2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicted by purple to green to yellow, and head motion is indicated by purple to red to yellow.	73
5.47	Image of Titanic.	74
5.48	The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).	75
5.49	Orange is vertical motion, and maroon is horizontal motion of the ear tip.	75
5.50	2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicted by purple to green to yellow, and head motion is indicated by purple to red to yellow.	76
5.51	Image of Tethys.	77
5.52	The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).	78
5.53	Orange is vertical motion, and maroon is horizontal motion of the ear tip.	78
5.54	2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicted by purple to green to yellow, and head motion is indicated by purple to red to yellow.	79
5.55	The comparison of a video with head and ear motion (left) vs. a video with just ear motion 5(right)	81
B.1	Large image of Big Red.	93
B.2	2D Head and ear motion with color changing with respect to time.	93
B.3	Graph of ear position, velocity and acceleration. Orange is vertical motion, and maroon is horizontal motion.	94
B.4	Graph of head position, velocity and acceleration through halter point. Orange is vertical motion, and maroon is horizontal motion.	94
B.5	2D Head and ear motion with color changing with respect to time.	95
B.6	Ear position, velocity and acceleration.	96
B.7	Ear movement measurements.	96
B.8	White Star	97

B.9	2D Head and ear motion with color changing with respect to time.	98
B.10	Ear position, velocity and acceleration.	98
B.11	Ear movement measurements.	99
B.12	2D Head and ear motion with color changing with respect to time.	99
B.13	Ear position, velocity and acceleration.	99
B.14	Ear movement measurements.	100
B.15	2D Head and ear motion with color changing with respect to time.	100
B.16	Ear position, velocity and acceleration.	100
B.17	Ear movement measurements.	101
B.18	2D Head and ear motion with color changing with respect to time.	102
B.19	Ear position, velocity and acceleration.	103
B.20	Ear movement measurements.	103
B.21	Big Benny	105
B.22	2D Head and ear motion with color changing with respect to time.	105
B.23	Ear position, velocity and acceleration.	106
B.24	Ear movement measurements.	106
B.25	2D Head and ear motion with color changing with respect to time.	107
B.26	Ear position, velocity and acceleration.	107
B.27	Ear movement measurements.	108
B.28	Chief	109
B.29	2D Head and ear motion with color changing with respect to time.	110
B.30	Ear position, velocity and acceleration.	110
B.31	Ear movement measurements.	110
B.32	2D Head and ear motion with color changing with respect to time.	111
B.33	Ear position, velocity and acceleration.	111
B.34	Ear movement measurements.	112
B.35	Mosfet	113
B.36	2D Head and ear motion with color changing with respect to time.	114

B.37 Ear position, velocity and acceleration.	114
B.38 Ear movement measurements.	114
B.39 2D Head and ear motion with color changing with respect to time.	115
B.40 Ear position, velocity and acceleration.	115
B.41 Ear movement measurements.	116
B.42 2D Head and ear motion with color changing with respect to time.	120
B.43 Ear position, velocity and acceleration.	120
B.44 Ear movement measurements.	120
B.45 2D Head and ear motion with color changing with respect to time.	121
B.46 Ear position, velocity and acceleration.	122
B.47 Ear movement measurements.	122
B.48 2D Head and ear motion with color changing with respect to time.	123
B.49 Ear position, velocity and acceleration.	124
B.50 Ear movement measurements.	124

List of Tables

2.1	RGB and HSV comparison of Figure 2.8.	11
4.1	High and low values of HSV used to filter the halter. All values between the values listed below become white all others become black.	24
4.2	High and low values of HSV used to filter the halter.	28
4.3	All of the distances between what the computer determined and the author determined as the ear tip, in pixels, for three random video frames from seven horses. Overall average distance is 4.7 pixels	35

List of Abbreviations

BRIEF Binary Robust Independent Elementary Features

CERT Computer Expression Recognition Toolbox

FACS Facial Action Unit Coding System

FAST Features from Accelerated Segment Test

FPS Frames per Second

HGS Horse Grimace Scale

HSV Hue, Saturation, Value

IACUC Institutional Animal Care and Use Committee

OpenCV Open Source Computer Vision Library

ORB Oriented FAST and Rotated BRIEF

OS Operating System

RGB Red, Green, Blue

SIFT Scale-Invariant Feature Transform

SURF Speeded-Up Robust Features

SVM Support Vector Machine

VT Virginia Tech

Chapter 1

Introduction & Review of Literature

The well-being of horses is important in many scenarios including therapeutic riding, racing, dressage, eventing, show jumping, polo, hunting, or just personal enjoyment among other activities. Knowing if a horse is in pain is difficult because horses are prey animals that have mechanisms in place that minimize the display of pain so that they do not become easy targets for predators [1].

There are many things that cause pain or stress in horses, including improper saddle fit, inadequate care, laminitis, lameness, illness, castration, and colic, among others. Horses are also transported all around the world and their well-being during transport is of paramount importance. Stewart, et al.[2] and Tateo, et al.[3] looked at the effects and behaviors of horses being transported by air and by land respectively.

The ability to remotely and automatically monitor horse well-being by exploiting image processing techniques will create opportunities to improve the welfare of horses. The work presented here looks at exploiting image-processing approaches to track facial features of horses in videos. This work develops a tool that can be used in the future for development of algorithms for automatic detection of pain in horses, whether during transportation, sporting events, or post-operative monitoring, among myriad other applications. This thesis details development of ear identification and tracking through video, head motion through halter identification and tracking, as well as a pilot study of eye detection. To obtain data to develop the techniques eleven horses were filmed of their left side resulting in 25 film clips used for demonstration of the techniques developed. A variety of different colored horses and breeds were used for this work to make as applicable as possible to the general class of horses and ponies. In the filming horses were led to a covered location, and all horses were required to wear a purple halter to utilize automatic color filtering in the videos.

All research was done with approval from the Virginia Tech Institutional Animal Care and Use Committee(VT IACUC), under the IACUC protocol number 18-022. All videos of horses were taken with the permission of the owners. The horses were left in the owner's care after videoing.

Image processing techniques were utilized from OpenCV, short for Open Source Computer Vision Library. OpenCV is a library of computer vision algorithms programmed by others for python, C++, Java and other programming languages. A few of the libraries used for horse facial feature detection and tracking are Canny edge detection; Hue, Saturation, Value

color filtering; and contour finding. Along with OpenCV, Python Libraries, and author generated code was used for the final code provided in Appendix C.

This thesis includes an overview of the literature on non-invasive detection of pain in horses and other animals, a brief overview of human facial feature detection in images and image processing applications on animals in Chapter 1. Chapter 2 covers background and fundamental information on image processing techniques utilized in this work to help readers understand terminology and approaches. Methods that were tried and not used in the final code, are discussed in Chapter 3 for future reference for other researchers. Chapter 4 details the methodology used and the algorithms developed for automatic detection and tracking of horse features. Chapter 5 discusses the data collection used in this work, and results. Chapter 6 is a conclusion with recommendations for future work.

1.1 Noninvasive Detection of Horse Pain

There has been research done on noninvasive detection of pain and stress in horses. Recently there has been work on developing a Horse Grimace Scale(HGS) by Dalla Costa et al [4] that is a noninvasive pain scale. This is based on six indicators on the face: ears backwards, orbital tightening, tension above the eyes, strained chewing muscle, pronounced chin, and strained or flattened nostrils [4]. Glerup et al. [5] looked at similar facial features, in a separate study around the same time, using two different pain stimuli. In the pain face study Glerup et al. looked at the same six indicators, but also noted the time to drowsiness in horses.

There have been other pain scales created that look at other indicators of pain. Glerup et al. [6] did a review of the different pain scales and proposed an Equine Pain Scale that combines all of the pain scales up to that point. The Equine Pain Scale looks at pain face, gross pain behavior, activity level, position in the stall, posture/demeanor, weight bearing, head position, attention towards the painful area, interactive behavior, and appetite scored between 0-4, 0 being no pain and 4 for display of high pain.

Dalla Costa et al. [7] did a pilot study to see if different emotional states affect the HGS. In the study they exposed each horse to four different conditions to evoke different emotion. They exposed each horse to a novel environment, grooming, anticipation of food reward, and a startling test. Seven healthy pain-free horses were used. Dalla Costa et al. concluded that emotional state did not change the HGS score, but the test needs a larger sample size.

Dyson et al. did two studies using the HGS to assess pain in ridden horses [8, 9]. Lameness was successfully determined in still images. In the 2018 study [9] described 24 markers on the whole body to determine the pain level of the horse.

All of the studies done on pain indication on the horse's face has been done looking at still images except for [9] which looked at videos. Regardless of whether the study looked at still

images or videos they were all looked at and scored by people.

This is where an automatic system for detection of pain or stress in horses in video and camera feeds, and the opportunity to develop a system using image processing approaches holds tremendous potential to improve animal welfare. Image processing algorithms such as there used in human facial recognition, and for guidance of autonomous vehicles, have the potential to be utilized for horse head feature detection and tracking. Horse head feature detection and tracking is an initial start towards the goal of automatic detection of pain and stress.

1.2 Recognition of Human Facial Features in Photos and Videos

This section details a broad overview of application of image processing for detection of human facial features in images. Human facial recognition has been researched for a long time. There are several ways of finding a face in the frame. One is Gabor filtering used in [10, 11, 12]. Another way of finding a face is active appearance models used in [13]. All of the mentioned research uses Support Vector Machines (SVM) to classify the differences in faces.

SVM creates a high dimensional plane that separates the different classes. SVM requires training to know the details of a class. In order to train the SVM to find faces, both pictures of human faces and pictures without human faces need to be supplied to the algorithm. The more images supplied to the algorithm the better classification. The better classification the less false positives and negatives there will be. SVM looks for the greatest distance between the positive and negative training data and creates a plane there to separate the data. Figure 1.1 [14] shows a 2 dimensional example of this where the white dots are one class and the black dots are another class and the red line is the separator between classes. SVM classification does not have to create linear planes; the image on the left of Figure 1.1 shows an example of a non-linear plane.

In 2011 Littlewort, et al. [15] created a program called the Computer Expression Recognition Toolbox (CERT). CERT can provide real-time feedback on 19 facial actions from the Facial Action Unit Coding System (FACS), intensity of smiles, 3D head orientation and the x, y position of 10 facial feature points. CERT's face detector was trained on the CMU+MIT dataset. The facial action unit detector was trained on the Cohn-Kanade, Ekman-Hager, M3, Man-Machine Interaction and 2 non-public databases. The smile detector was trained on a subset of 20,000 images from the GENKI dataset.

There are a lot of databases that have been created for human faces. Cole Calistra lists 60 databases that can be used to train facial detection, and recognition [16]. There are also databases for facial expression recognition [17]. These databases have the ground truth pro-

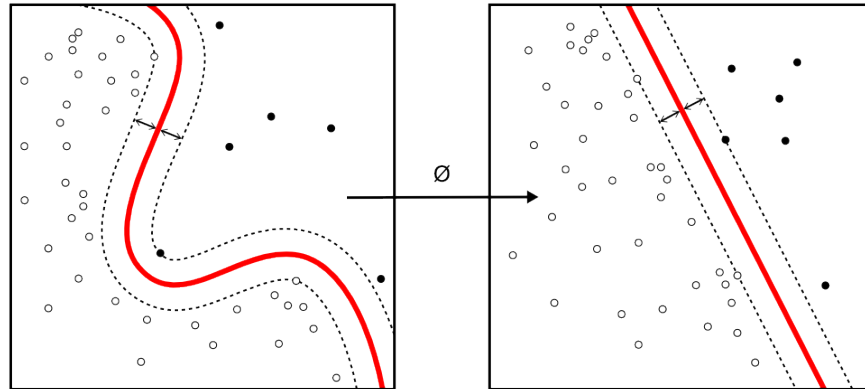


Figure 1.1: 2D example of a plane created to separate two classes. A non-linear plane shown on the left and a linear plane on the right.

vided either the emotion or the action unit (AU) displayed. The only database specifically for horses is Arnd Bronkhorst Horse Photos, Horse Image Database [18]. The Arnd Bronkhorst Horse Photos, Horse Image Database is a database created for selling horse photographs. No database of horse images created for academic research was found at the writing of this thesis.

1.3 Applications of Image Processing on Animals

A lot of work has been done in computer vision for facial recognition on humans, but not much has been done on horses or animals in general. The closest work on horses uses Haar-like features to detect whole horses in still images [19]. Uddin and Akhi were able to achieve a success rate of 63% with 74 false positives. The horse detector was trained to only find the side view of horses and took the authors a long time to go through the many thousands of images. To train the horse detector the horse had to be manually segmented from the background, and resized. Then two text files containing all of the image locations needed to be created, one for the positive images and one for the negative images. The negative images can be any image without a horse. In 2017 Lu et al. [20] looked at automating the scoring of pain on sheep. The authors of the paper used a database of 480 sheep images created by Yang et al. [21]. The Yang et al. dataset manually labeled the bounding boxes and 8 landmarks on the sheep faces of 600 sheep faces. To get more training images Lu et al. clipped the sheep faces rotated them and placed them on random backgrounds. creating a final dataset of 5000 images. In the paper they used localized Histogram of Oriented Gradients for feature description and Support Vector Machine to classify the features.

Another paper that that uses computer vision for pain detection on animals is "The Rat Grimace Scale: A Partially Automated Method for Quantifying Pain in the Laboratory Rat via Facial Expressions" by Sotocinal et al. [22]. For the paper Sotocinal et al. created

a program to called Rodent Face Finder to aid in the study of the Rat Grimace Scale. The program uses boosted cascades of Haar classifiers to detect the ears and eyes of the rodents. To train the program the authors used 500 cropped images of ears and eyes as positive examples and non-face containing frames and unrelated images as negative training examples. The Rodent Face Finder was used to find frames in a video where the rodents face is visible, but the pain score is determined manually by trained researchers. The rodents were contained in a small acrylic box where they were allowed to roam free.

This thesis lays the foundational work for tracking ear motion, head position and angle, and eye status in video of a side view of a standing horse facing to the left. The reason for focusing on those three are the majority of papers mention ear motion [2, 3, 4, 5, 8, 9], head motion [3, 5, 8, 9], and eyes or time to time to drowsiness [2, 5, 8, 9]

Chapter 2

Image Processing Background

This chapter discusses the background of image processing and gives an overview of the algorithms used to detect the features on a horse.

2.1 Python Programming Language

The Python programming language was chosen for this thesis because of its easy of use, large number of libraries and large community of users. Python is an interpreted programming language that has many libraries. Python focuses on human readability which makes it easy to learn, read, and understand other's code[23].

A library is a collection of prewritten code algorithms that do specific tasks. Some examples of functionality that the Python libraries add are, differentiation, integration, linear algebra dot and cross products, matrix math, plotting, image manipulation and much more. Without the libraries the size of the code and the time to write the program would be much larger.

Python also has a large community of helpful users that are constantly creating tutorials and example code that they make available to everybody. Since there are so many people using Python and sharing their examples and troubles they have experienced finding solutions is much easier than only reading through the library definitions and trial and error. The library definitions are always a good place to start but not always the most helpful, some are written very well and have helpful examples, while others are very basic and only have minimal information. The community is also adding to libraries and creating new ones that expands on the functionality of python.

Python is platform independent, which means it can be installed on Windows, Macintosh, and Linux. The code can also be written on one Operating System (OS) and run on a different OS. Even though Python is platform independent not all of the libraries written for Python are platform independent. The code in this thesis has been written to be platform independent to the best of the author's knowledge. The code has been written using Python version 3.5.2, OpenCV 3.2.0, Numpy 1.12.1, Matplotlib 2.0.2, was written on Linux Mint 18.1, and has not been tested on any other systems. The code is provided in Appendix C.

2.2 The OpenCV Library

OpenCV is short for Open Source Computer Vision Library [24], and is the main library used in this thesis. OpenCV is a collection of computer vision algorithms written in optimized C/C++ and is free for both academic and commercial use. "OpenCV has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android [24]." The library has more than 2500 optimized algorithms saving time writing code. There are more than 47 thousand users of OpenCV that makes it easy to find help when something is not working as expected. The following sections go into detail about computer vision and the OpenCV functions used to find the features on a horse's head.

2.3 Image and Video in Computer Vision

In computer vision, images are stored as a set of 2D matrices; one 2D matrix per color red, green, and blue(RGB), and videos are a sequence of images. Each element in the matrix represents a pixel in the image. The value of each element is between 0 and 255. In a grayscale image, black pixels are 0, white pixel are 255, and the numbers in between are shades between black and white. Figure 2.1 [25] shows a grayscale image of a vehicle and in yellow is how the computer sees an image, each element in the portion of the matrix is a number representing the grayscale value of the corresponding pixel in the image. The computer sees a group of numbers where we see a car mirror, that is why there is still a lot of work being done in computer vision, and why OpenCV is so important. A color image is similar to a grayscale image except it has three matrices all the same size, one matrix for each color red, green, and blue (RGB). Each element is still between 0 and 255 this time 0 is black and 255 is full color. When looking at the red matrix 0 is a black pixel and 255 is a red

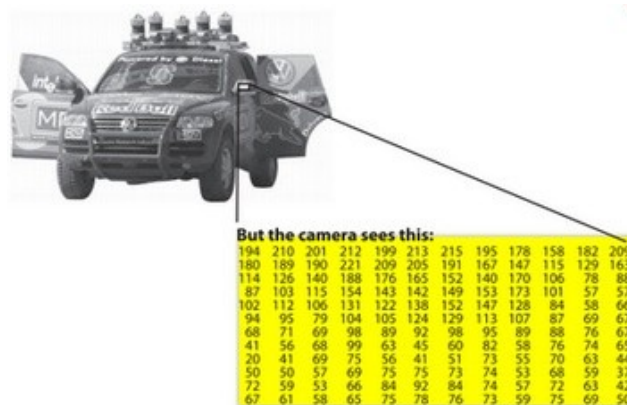


Figure 2.1: How humans see an image and the matrix, in yellow, of how computers and cameras see the image.

only pixel and the numbers in between are the shades of red. This is the same for the blue and green matrices. TVs, most electronics and Python's Matplotlib represent color images using the order red, green, and blue, but OpenCV represents color images in the order blue, green, red (BGR). This is important because the matrix order needs to be changed before displaying images in matplotlib or the image will look strange. Figure 2.2 from a blog on the Chinese Software Developer Network [26] shows what an image might look like if the RGB isn't switched, the left image shows how the colors are different if not switched and the right image is the original.

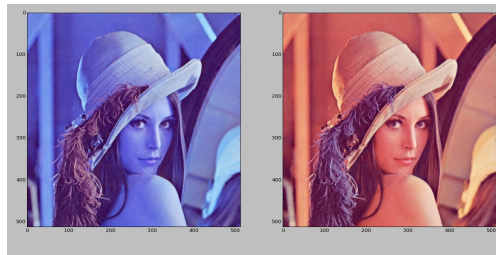


Figure 2.2: The left image show an OpenCV image displayed with Matplotlib without switching to RGB order. The right image is the original image.

Image processing requires a lot of computer resources therefore it is common to work with grayscale images because they are 1/3 the size of color images. Since the images are smaller the filters and functions run much quicker than they would on a color image.

Videos are treated similar to images. Once a video is loaded by the program the first frame is grabbed. The frame is processed as an image when the program is done processing the image the next frame is grabbed and the process continues until the video is finished.

2.4 Linear Filtering

Filtering is a common operation in image processing. A few examples of filtering are blurring, sharpening, and finding edges in images. The Canny edge detection discussed in the next section uses a Gaussian filter to blur the image and reduce noise. Canny edge also uses both the X and Y direction Sobel filters to find the edges. A filter is a fixed size matrix, commonly square, know as a kernel. Figure 2.3 shows the X and Y direction of the Sobel filter as an example of a kernel. To filter an image the kernel is run across the each image starting with the top left pixel which is index (0,0). All of the neighboring pixels are multiplied to each element in the kernel then added together. After the summation the value is stored in the same location in a new image. The equation for this is shown in Equation 2.1 where $g(i, j)$ is the resulting pixel at location (i,j). $f(i+k,j+l)$ is the pixel location of the original image plus

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figure 2.3: Example of a kernel with the X-direction Sobel filter on the left and Y-direction Sobel filter on the right.

the index of kernel, and $h(k,l)$ is the pixel location of the kernel.

$$g(i, j) = \sum_{k,l} f(i + k, j + l)h(k, l) \quad (2.1)$$

Figure 2.4 shows a simplified image as the computer sees images. For example if we run the X direction Sobel shown in left of Figure 2.3 on pixel in the top row first column or $f(0,0)$ the results would be stored in $g(0,0)$ and Equation 2.1 would look like Equation 2.2 and that pixel would equal Equation 2.3 also shown in matrix form in Equation 2.5.

$$f(i, j) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Figure 2.4: Simplified example of an image.

$$g(0, 0) = \sum_{k,l} f(0 + k, 0 + l)h(k, l) \quad (2.2)$$

$$g(0, 0) = (1 \times (-1)) + (2 \times (0)) + (3 \times (1)) + (5 \times (-2)) + (6 \times (0)) + (7 \times (1)) + (9 \times (-1)) + (10 \times (0)) + (11 \times (1)) = 8 \quad (2.3)$$

$$g(0, 0) = \begin{bmatrix} 1 \times (-1) & 2 \times (0) & 3 \times (1) & 4 \\ 5 \times (-2) & 6 \times (0) & 7 \times (2) & 8 \\ 9 \times (-1) & 10 \times (0) & 11 \times (1) & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} = \begin{bmatrix} 8 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \quad (2.4)$$

Skipping ahead to $G(3, 0)$ and ignoring the results of the first three positions you will notice that the filter over hangs the image shown in Figure 2.5, so a border needs to be added around the image that is larger than the filter. In the example above the $(0,0)$ point of the kernel was on the top left, but unlike an image that is not always the case, the $(0,0)$ point can be the center, bottom left or any position in the kernel. Since the $(0,0)$ point of the kernel can be anywhere and is commonly the center of the kernel a border needs to be added all the way around an image. OpenCV has 5 built in functions to add borders around the image shown in Figure 2.6 [27]. If no border is specified then the default border is used, which is REFLECT_101

$$g(3,0) = \begin{bmatrix} 1 & 2 & 3 & 4 \times (-1) & ? \times (0) & ? \times (1) \\ 5 & 6 & 7 & 8 \times (-2) & ? \times (0) & ? \times (2) \\ 9 & 10 & 11 & 12 \times (-1) & ? \times (0) & ? \times (1) \\ 13 & 14 & 15 & 16 & ? & ? \end{bmatrix}$$

Figure 2.5: The overhang of a kernel when it is near the borders of an image.

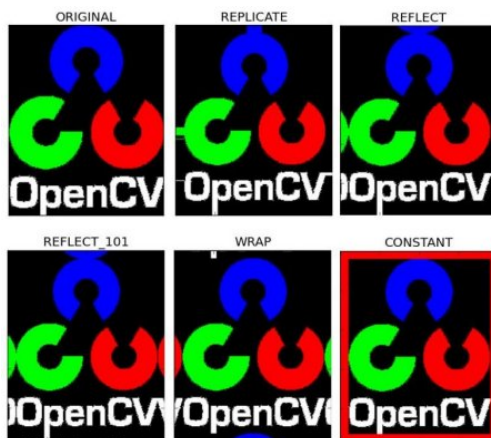


Figure 2.6: Visual representation of the different borders in OpenCV.

2.5 Hue, Saturation, Value (HSV) Colorspace

A color space is a specific way of organizing colors. Most people are taught as kids about the red, blue, green or yellow colorspace, when given red, blue and green, or red blue, and yellow paint and asked to mix the colors to make other colors. Printers use cyan, magenta, yellow, and black to create color images. There are a lot more colorspace each with advantages and disadvantages. The hue, saturation, value (HSV) colorspace is used in this thesis. Hue refers to the pure color eg. red, green, blue, magenta, etc. Saturation is how white the color is. For the color green a saturation of 255 is green and 0 is white. Value is how dark the color is, 0 is black and 255 is the color. Figure 2.7 from Wikipedia [28] shows a visual representation of the HSV colorspace where each hue, or color, is a slice of the cylinder. Saturation moves radially in and out of the cylinder between color and white. value moves up and down between the color and black.

HSV is easier to think of how colors change because you start off with a color like red, orange, or teal. Then white and black gets mixed in to get the tints, tones and shades of that color. Rather than thinking of how much red, green, and blue to mix together to get a different color then its tints tones and shades. It is much easier to think of orange as hue 28 saturation 100 value 100 than red 255 green 124 blue 0. An example of why HSV is easier to think of and filter is shown in Figure 2.8 also from Wikipedia [29]. The figure shows

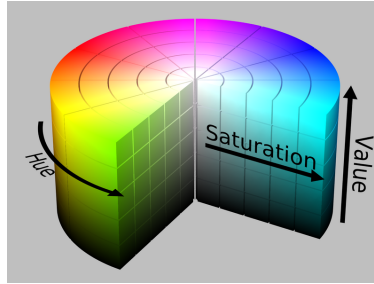


Figure 2.7: Visual representation of HSV, where each thin slice represents a hue

two similar colors that you might want to filter out from the rest of the colors in an image. Changing from one color to the other it can be seen that the amount of red goes down by 31, the green goes up by 24 and the blue goes up by 59. Table 2.1 includes the HSV values of the two previous colors, notice that the hue is the same for both colors. All that is need to find the colors is filter the hue of 28 and putting a range on the saturation and value. Even if an object has a shadow the hue of the shadow is the same or close to the same number as the original object whereas in RGB all three numbers change at different rates each time the color is chaged making HSV much easier to filter than RGB.



Figure 2.8: Add caption.

Table 2.1: RGB and HSV comparison of Figure 2.8.

	RGB	⇔	HSV	RGB	⇔	HSV
Red / Hue	217	⇔	28	186	⇔	28
Green / Saturation	118	⇔	85	132	⇔	54
Blue / Value	33	⇔	77	85	⇔	73

2.6 Canny Edge Detection

”Edges in gray-level images can be thought of as pixel locations of abrupt gray-level change. [30]” Canny edge detection is named after the creator John Canny, who in 1986 created a way of computationally detecting edges in an image. The edge detector works by first

running a Gaussian filter to remove any noise by blurring the image. After the Gaussian filter two Sobel filters are run one in the X direction G_x and the other in the Y direction G_y both shown in Section 2.4 Figure 2.3. The X direction Sobel finds all the vertical edges and the Y direction finds all of the horizontal edges. The other edges are detected by using the Equation 2.5 to extract the other angles from the X and Y Sobel.

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \quad (2.5)$$

In OpenCV Canny edge detection uses two values called maxVal and minVal. Any edge with an edge gradient, determined by Equation 2.6, greater than the maxVal is determined to be an edge. Any value below minVal is for sure a non-edge and is removed. The lines in between are determined based on connectivity. A line that is connected to a sure edge is considered to be an edge any other is discarded. This is shown in Figure 2.9 [31] where edge A is above the maxVal so it is kept as an edge. Edge B is above the minVal but it is not connected to a for sure edge so it is discarded. Since edge C is connected to edge A which is above maxVal edge C is kept as an edge.

$$G = \sqrt{G_x^2 + G_y^2} \quad (2.6)$$

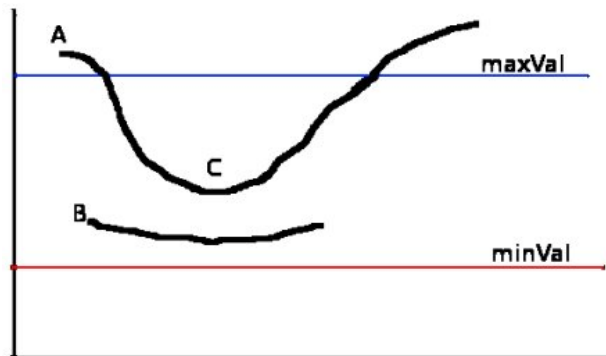


Figure 2.9: A visual representation of the minVal and maxVal. 'A' is an edge because it is above maxVal, 'B' is not an edge because it is below the maxVal and not connected to an edge above, but 'C' is an edge because it is connected to 'A' which is above maxVal.

2.7 Contours

"A contour is a curve joining all the continuous points along the boundary, having same color or intensity [32]."

Contours in OpenCV have several features that help filter, sort and locate a contour. The features of a contour are: moments that can be used for finding the centroid; area; perimeter; approximation (shape comparison); convex hull (corrects convexity defects); Straight Bounding Rectangle; Rotated Rectangle; Minimum Enclosing Circle; Fitting an Ellipse; and Fitting a Line.

Contours also have many properties, similar to features that also aid in filtering and sorting. Examples of the properties are: aspect ratio, extent, solidity, equivalent diameter, orientation, mask and pixel points, maximum value, minimum value and their locations, mean color or mean intensity, extreme points.

Chapter 3

Pilot Study for Head Tracking

This chapter gives background and information on initial tracking methods that were explored but did not provide results desired. However, they informed the direction of the work and represents other possible paths that maybe useful in future studies.

3.1 Grab Cuts

Grab cuts is a way of defining what is foreground and what is background in an image. Grab cuts was abandoned because it required a human to define what is foreground and what is background, and the goal of this project is to be as automatic as possible. Grab cuts was the first attempt at differentiating the horse from the background, but grab cuts did not remove all of the back ground and remove part of the horse. There are no images of the results because this was done when the author had permission to take and use videos/images for testing and development, but did not have permission to publish the videos/images. The videos that were taken early in the research. The early images helped guide some of the structuring used in the publishable images. Grab cuts would work better on the new data, but still requires a person to specify what part of the image is horse and what is background.

3.2 Dots on The Horse

The original idea was to do pain detection with out adding anything to the horse. Just simply point a camera at the horse and have the computer determine if the horse is in pain. With computer vision that turned out to be very difficult. Finding features on a horse head was proving to be difficult and time consuming. It was suggested to simplify the problem then build from there. So instead of looking for features on the head then tracking the features through multiple frames in a video, dots were placed on the horses head. Brightly colored dots where chosen to highly contrast from the horses fur. HSV filtering was used to find the dots then blob detection was used to track the dots.

The first video that was taken used three different colored half inch yard sale stickers pink, green, and orange. The pink dots were used for the first experiment just because that is the color that we happened to put on the ear and the neck. some code was written to calculate

and graph the change in ear movement. The horizontal, vertical, and linear distance were all graphed separately. The dot on the ear would disappear from the camera when the horse rotated its ear forward. When the computer only saw one dot all the values were set to zero. Zero was used, because the dots could never physically have a distance of zero and the value of zero was a drastic change making it clear what was happening.

The dots were not used because they required a person to place them on the horse. The stickers might change the horse's behavior because it may introduce a slight discomfort especially when stuck to the ear. The endurance of stickers ability to stick to fur could also complicate long observations. For the previous reasons the stickers were abandoned, but led to the idea of using purple colored halters. Halters rest on close to the same spot for every horse so it provides a good landmark to work from and horses are already use to wearing them so they should have little effect on horses' behavior.

Again there are no images for this section due to permissions.

3.3 ORB (Oriented FAST and Rotated BRIEF)

Oriented FAST and Rotated BRIEF (ORB) is a combination and improvement of FAST (Features from Accelerated Segment Test) and BRIEF (Binary Robust Independent Elementary Features). ORB is an alternative to SIFT (Scale-Invariant Feature Transform) and SURF (Speeded-Up Robust Features). ORB is a good alternative to SIFT and SURF because it has the same performance in feature match, but is computational more efficient, and is not patented [33].

The following sections describe the three attempts to improve and use ORB feature tracking. It was put aside to try other halter trackers. In each frame ORB found different points to track so it was inconsistent for tracking specific points on the halter. ORB still has potential to be used in the future, because it did a good job of matching points between frames.

3.3.1 Attempt 1 of 3 for Halter Tracking Through ORB Feature Matching

The first attempt ORB was used on the halter binary image after HSV color filtering. The point found in the first frame were not the same points in the second frame. Figure 3.1 shows the 20 best matches on two images side-by-side. The halter is white in the image. On the left is the first image and on the right is an image where the halter has move a little bit. The lines connect the points that the program thinks are the same in each image. Because the halter has only moved slightly all of the lines should be close to parallel. Notice however that there are line that cross and connect points at different spots on the halter. The lines that cross are false matches that need to be filtered out. It makes sense that the program is

doing that because the feature descriptors do not have enough information to create better matches.

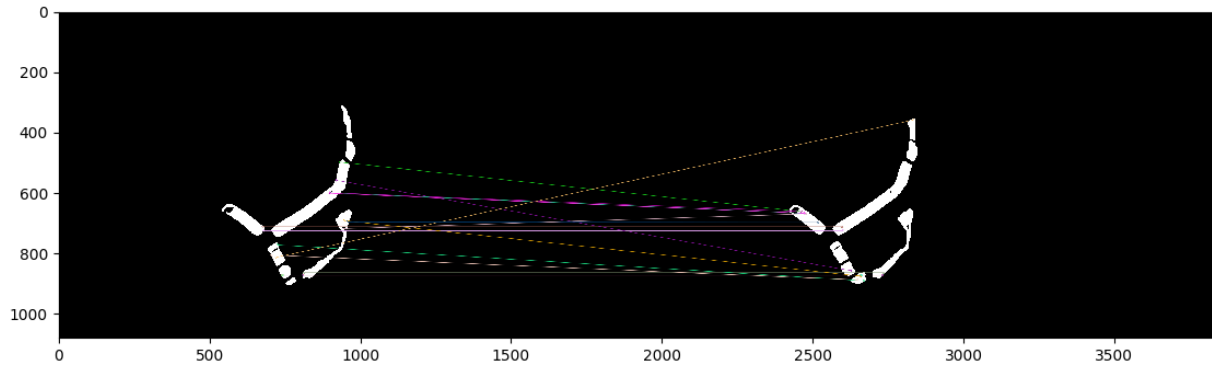


Figure 3.1: ORB halter point matching attempt one. Lines should connect the same point on each halter.

3.3.2 Attempt 2 of 3 for Halter Tracking Through ORB Feature Matching

The second attempt uses the grayscale version of the image but uses the binary image as a mask so the program only finds looks for features in the white areas of the mask which in this case is the halter. Figure 3.2 shows the 20 best matches. There is less crossing but the lines still connect different spots on the halter.

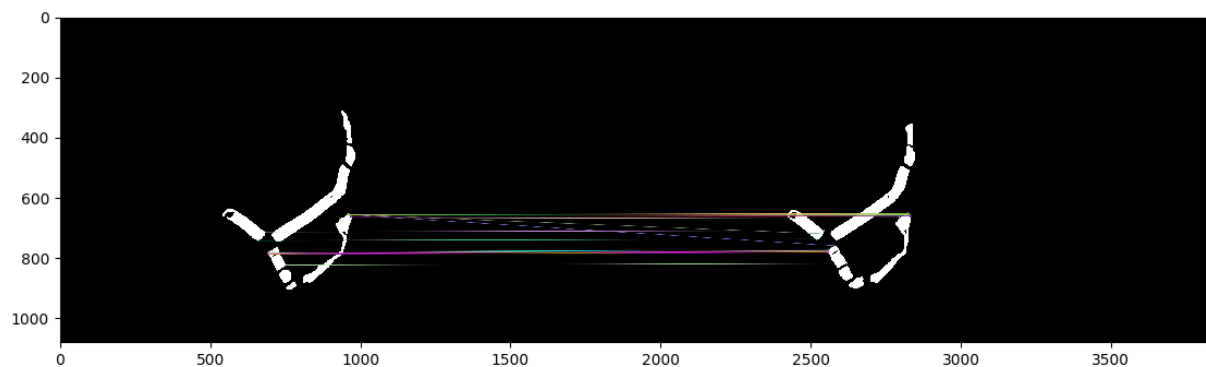


Figure 3.2: ORB halter point matching attempt two. Lines should connect the same point on each halter.

3.3.3 Attempt 3 of 3 for Halter Tracking Through ORB Feature Matching

Uses the binary image to detect good features on the halter and store their location. Then the program uses the location to calculate the feature descriptors on the grayscale version of the image. This gives excellent results that can be seen in 3.3. All of the lines are parallel and connect to the same point on the halter in both images.

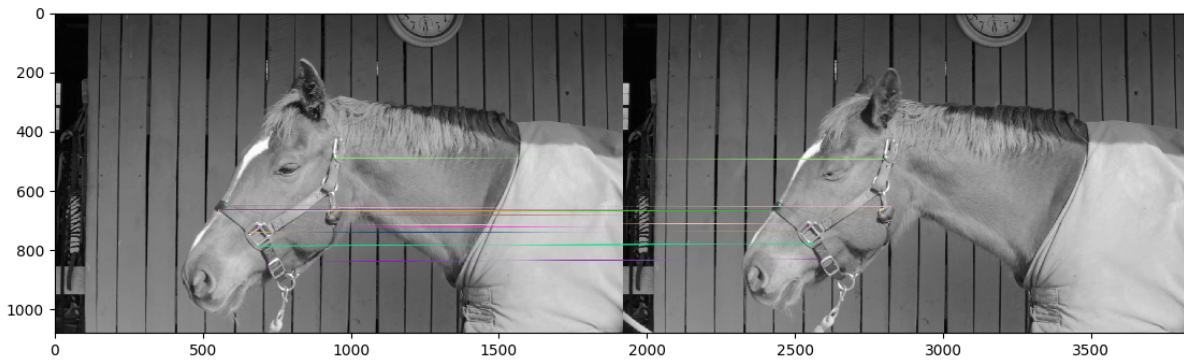


Figure 3.3: ORB halter point matching attempt three. This shows the best results where the lines connect to the same point on each halter.

3.4 Halter Tracking using Hough Lines

Hough lines was tried to track the halter, it was later replaced by tracking contours. The way Hough lines was used to find all of the lines in the HSV filtered image of the halter. The lines were found using the parameters of 1 pixel resolution for rho, 1 radian resolution for theta, and a threshold of 40, found by trail-and-error. Hough lines use polar coordinates to describe the lines therefore it is easy to calculate the needed angle. Hough line transform finds more lines than are needed so the lines are grouped based on angles. To group the angles the lines are sorted from smallest to largest angle. A difference is taken between each angle if the difference in angle is less than 0.1 radians the angle and distance is stored in a array. If the difference is greater than 0.1 all the lines in the temporary array are averaged and the averaged line is stored in a new array while the temporary array is cleared to start the process over again on the next group of lines. Once the lines are grouped each group is averaged to find one line per group. Since there is 2-3 lines instead of 20+ lines the angles can tracked much easier. The intersection of the lines can also be found giving a good reference point on the head that can be tracked without using stickers on the horses heads and necks.

In the end tracking contours was used because it did a better job of finding the parts and orientations of the halter parts.

Chapter 4

Methodology

This chapter details how each of the two elements on the horse's head are identified and tracked, both head motion (through halter tracking), and ears, additionally, a pilot approach for eye detection is presented. Head tracking through the halter was done first since the halter is a known landmark that stays relatively still with any head motion. Once the halter was identified, it was used as a reference in the algorithm to narrow regions in the image for ear and eye location, respectively.

4.1 Experimental Testing Procedure

The videos were taken in a way to replicate a typical field scenario as closely as possible. The horses were taken right out of the pastures and brought to covered grooming area by the barn. except for purple halter no other test parameters were instituted to this typical field scenario. horses brought up and held by lead rope which is common and familiar to the horses in this location.

The videos were taken on the morning of Wednesday, March 7th 2018 at a horse boarding and training facility in Blacksburg, Virginia. The weather was overcast with light snow that changed over to sunny skies. The videos were recorded on a Canon 60D with a Canon EF-S 18-55mm lens. The camera was mounted on a tripod.

At least two videos were created of eleven different horses. One of the two videos was recorded at a resolution of 1920 by 1080 pixels at 29.97 frames per second(fps). The second video was recorded at a resolution of 1280 by 720 pixels at 59.94 frames per second. The recordings were made in two different resolutions and frame rates to test if it is more important to have a high resolution or high frame rate. Both types of recordings were used in this thesis, and the determination of which is better high resolution or high frame rate did not matter in this study, and was left for future work.

Each video has at least 1 minute of the horse's head in frame. Some videos start before or end after the horse is out of frame, this is to get video of the background to be used for background subtraction. The videos have been edited into shorter clips to remove people and extremes in head position that the program can not yet handle. A purple halter was placed on each horse as described in Section 4.3.

Each horse was given a new name to make identifying the horse easier in this thesis, but keeping personal identifying information about the horse anonymous. The horses names and image of each horse is shown in Figure 4.1.

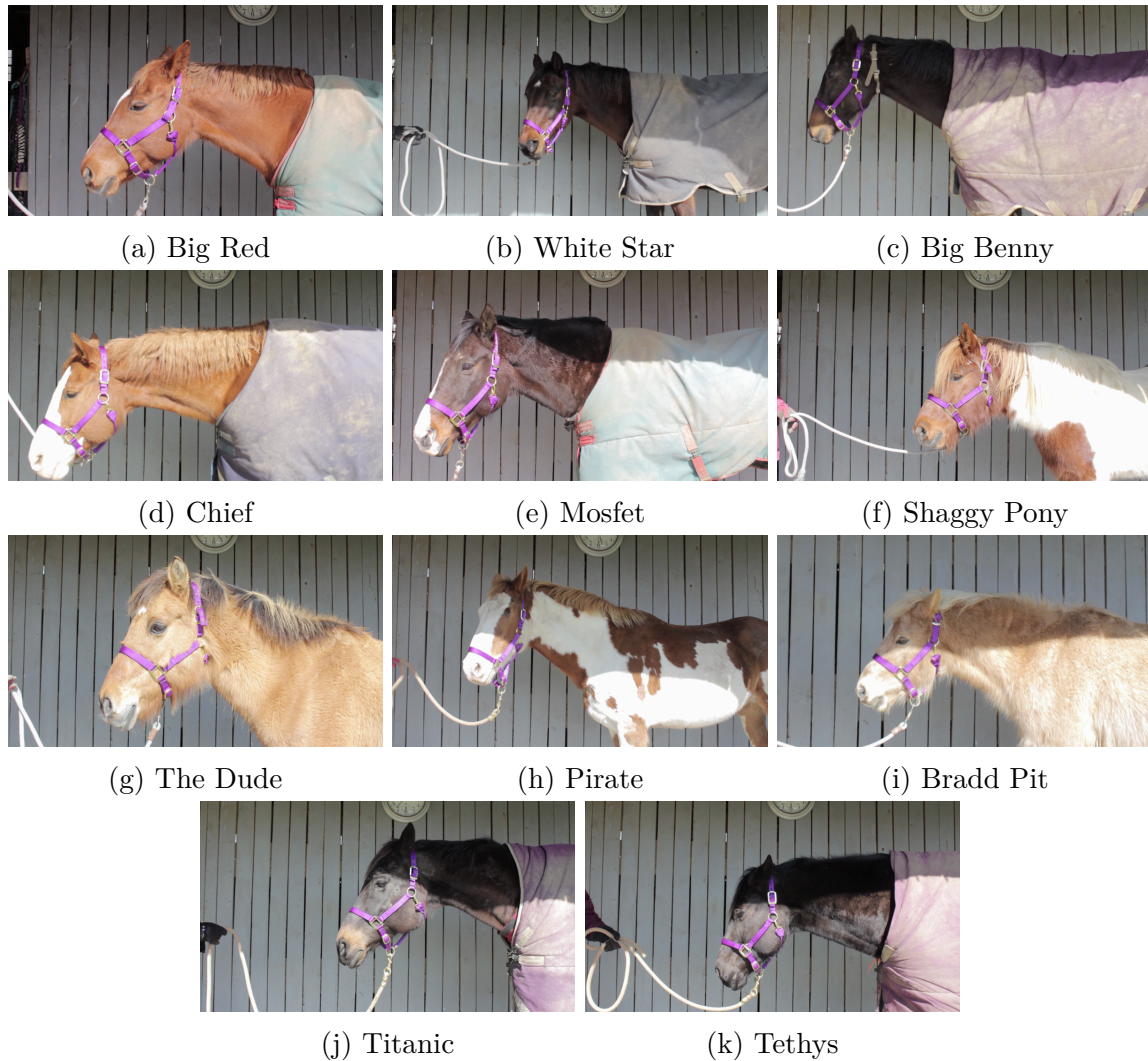


Figure 4.1: Images of the horses recorded for this thesis and their research names (not real names).

4.1.1 File Naming Conventions

The videos recorded for this thesis all use the default name that was given by the camera, "mvi_####.mov". When clips were cut from the original videos a new naming convention was used. Each clip name starts with the 4 digit number given by the camera, followed by

edit and a two digit number to denote multiple clips from the same video. That is follow by the size of the video frame in width then height. All elements in the name are separated by underscores. An example is '4893_edit00_1920_1080.mov', original video is mvi_4893.mov this is the first clip from the video and it is a 1920 by 1080 pixel frame size. Comma separated value(CSV) file and figures use the same name as the clip they represent with addition information added to the end. The naming for figures is clip_name_Figure_1.png for the 2 dimensional graph of head and ear movement; clip_name_Figure_2.png is for the ear motion, velocity, and acceleration; and clip_name_Figure_3.png is for the head motion, velocity and acceleration graphs.

4.2 Overview of Detection and Tracking Code

Video of horses that were collected were run through the program Horse Head Tracker (horse_head_tracker.py) Figure 4.2 shows a flow chart of the complete program, and begins at the top right with video_frame, that indicates the raw data from the camera. The square blocks are the functions of the program with the name of the function as it is in the code. The parallelograms are inputs and outputs of the functions. The blue parallelograms are images, the white parallelograms are the other outputs such as (x, y) points, with the exception of bounding_boxes which is a list of points needed to create the bounding boxes, and csv which is a csv file. The large gray boxes highlight the section of the program that are discussed in the corresponding section below.

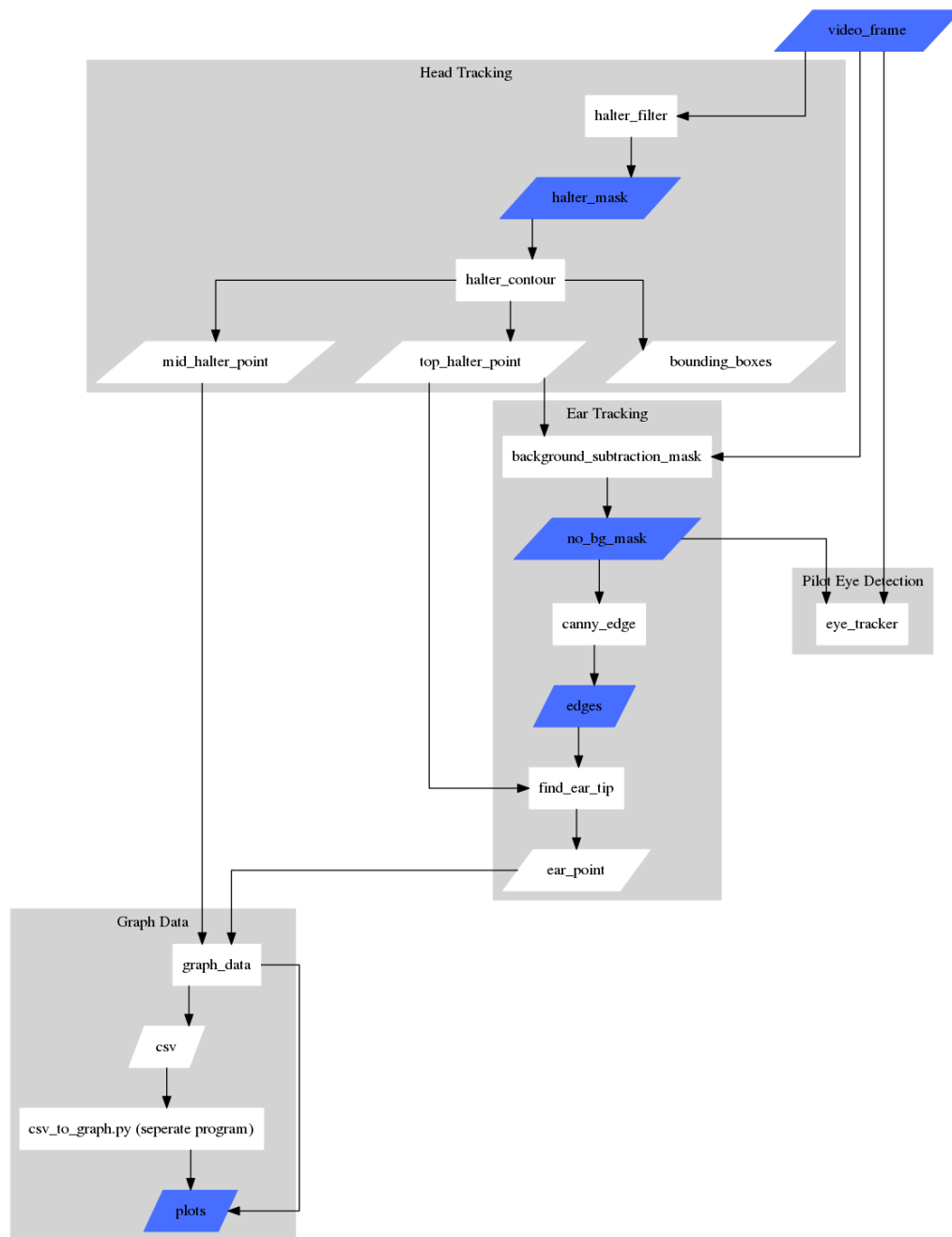


Figure 4.2: Flow chart of program

4.3 Head Tracking through halter identification

Domesticated horses are often wearing halters this was exploited to begin development of an approach for facial feature identification. The halter also moves with the head and provides a distinct object to track. In the approach horses are filmed wearing a purple halter as shown in Figure 4.4a. The reason purple was selected as the color to use is because it is an easy color to track using HSV methods because it is a different color than horses, as well as their normal surroundings. Purple is not blue like the sky, green like grass, or the natural color of horses. Red was also considered, but the reason for not choosing red is because the color wheel is continuous, but in OpenCV the color wheel is finite and red is at zero which means it is also at 360 degrees on the color wheel. If you unroll the color wheel into a line, this means red is at the beginning and end of the line shown in Figure 4.3 [34]. Since red is divided to each end of the line extra code is required to compensate, which makes it more difficult to use. There are also horses that are close to the color red that could also pose issues, an example of the color is a horse, named Big Red for this study, shown in Figure 4.4a. Therefore purple was selected as a distinct color different from all possible natural horse colors. After data collection it was discovered that purple is a common color for other horse accessories like turnout blankets, and horse lead ropes, which should be avoided or improved upon in future work.

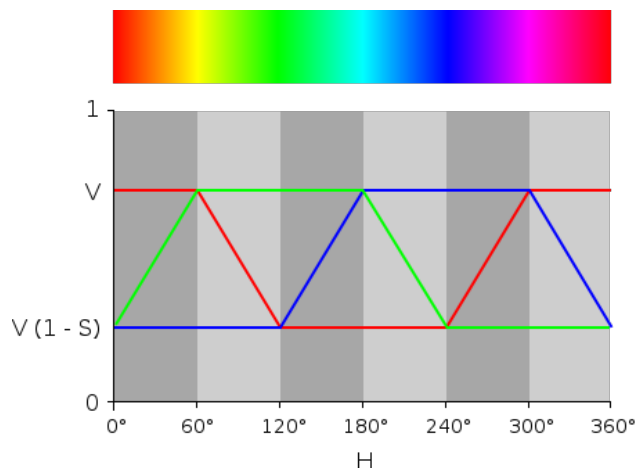


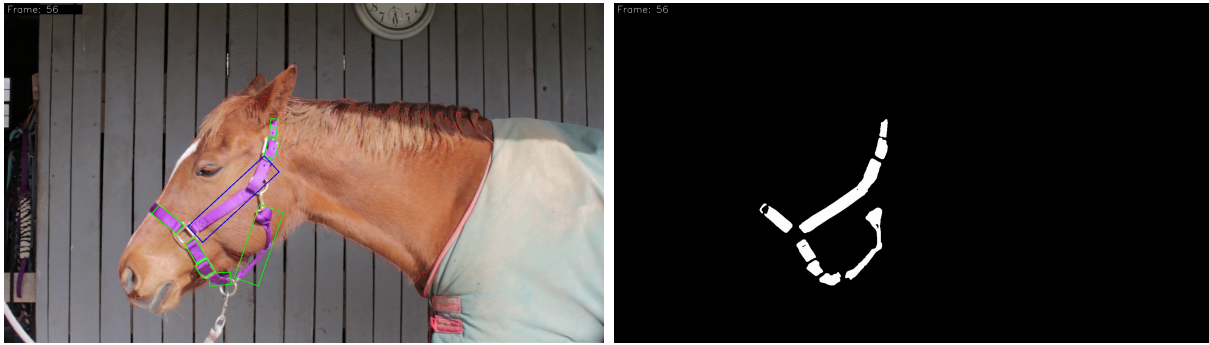
Figure 4.3: A graphical representation of RGB coordinates given values for HSV. The top color bar shows the separation of red. 0 to 360 degrees on the color wheel is represented by the values 0 to 255 in OpenCV.

In the horse images, the purple halter is filtered from the rest of the frame using the Hue, Saturation, Value colorspace (HSV) as described in Section 2.5. The Hue describes the color or location on the color wheel ranging from 0 to 255. The Value and Saturation describe how much black and white respectively the color has also ranging from 0 to 255 each. The HSV values used to find the purple halter in the frame are listed in Table 4.1. HSV filtering returns a black and white image like the one in Figure 4.4b, this image is commonly called a mask, because it is similar to masking off an area before painting except the black and white image is used to tell the computer which are important pixels and which pixels need to be ignored.

Table 4.1: High and low values of HSV used to filter the halter. All values between the values listed below become white all others become black.

	Low	High
Hue	133	160
Saturation	56	182
Value	74	255

After the black and white image is created (Figure 4.4b) all of the contours are found. Contours are a closed curves joining all the continuous points along a boundary, having same color or intensity[32] as described in Section 2.7. The contours are found in the image by using the `findContours()` function in OpenCV and using the flags `RETR_EXTERNAL` which only returns the extreme outer contour points ignoring any holes in the shapes and the



(a) Original image.

(b) HSV filtered halter.

Figure 4.4: A single frame of Big Red. On the left shows the original color image with the bounding boxes drawn, and the right image shows the halter after HSV filtering which is used to create bounding boxes.

CHAIN_APPROX_SIMPLE flag to remove all redundant points on the contour which saves memory and decreases processing time. Once all the contour points are found for the halter, a rotated bounding box is found for each contour. The bounding box is the smallest rotated rectangle that can fit around the contour. Each bounding box returns the center point, the width and height of the rectangle, and the angle that the box is rotated. The bounding rectangles are shown drawn, as green or blue boxes around the halter, on the original image in Figure 4.4a. The boundary boxes allow us to see what contours the computer is tracking, the was used to differentiate the box used for head tracking but has been replace by a blue dot on the center point of the bounding box.

As the bounding rectangles are created they are put into a list. After all the rectangles are entered into the list the order is reversed to sort all the rectangles from the top of the frame down. The boxes are sorted from top down so the top bounding box is always the first item in the list making it easier to tell the program where to grab the information to be used in ear tracking described in Section 4.4.

The bounding boxes are then resorted from left to right using the center points, to always make the left most box the first box in the list. The program can now grab the first box in list rather than determining which box in the list is the correct one. The left most bounding box, which is usually the nosepiece, shown in Figure 4.5 [35], of the halter when horse is facing left. This is the reason horses have to face left for the Horse Head Tracker code written for this thesis. The nosepiece is used to track the halter and by extension the head motion. This point is used because the left most point does not jump around while the horse turns or nods its head and returned the best results. Previously the third bounding box from the top was used for head position tracking, but was changed because the amount and order of the contours changed as more or different parts of the halter became visible or blocked.



Figure 4.5: The parts of the halter.

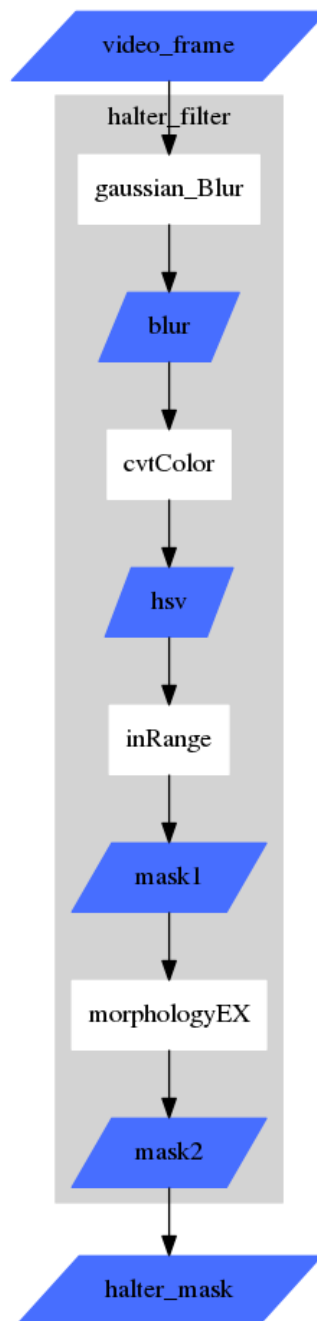


Figure 4.6: Flow chart of the function halter_filter

4.4 Ear Tracking

The halter location is utilized to find the left ear by finding the center point of the top halter bounding box is used to initially find the ear tip. Before the ear tip can be found, an outline of the head needs to be created; this is done by finding the Canny edges, as described in Section 2.6, of the head after the background is removed. The Canny edges were found without removing the background, but the results needed improvement so the background was removed before finding the Canny edges. The background is removed from the image to remove all unwanted information from the image and to make the edges of the horse more clear.

To remove the background, HSV color filtering is used to find the color of the horse. Now there are two different groups of HSV numbers, one to find the light horses, and the other to find the dark horses. A horse is considered dark if it looks black from a distance, examples of dark horses in this thesis are White Star, Big Benny, Mosfet, Titanic, and Tethys. Some adjustment may need to be made to accommodate other colors or color patterns, because a limited number of horses were used in this study. The HSV values are shown in Table 4.2, with the values for the light horses on the left and the values for the dark horses on the right. The program automatically determines if the horse is a light or dark horse by looking at the center point of the top halter bounding box, and looking at the HSV value of the pixel 20 pixels to the left of the center point. Figure 4.7b and 4.8b show the results of the HSV on a light horse and dark horse respectively. On the dark horses some of the horse's head is seen as background by the computer because of the shadows, but this is okay as long as the tip of the ear is not removed.

Table 4.2: High and low values of HSV used to filter the halter.

	Light Horse		Dark Horse	
	Low	High	Low	High
Hue	0	25	0	255
Saturation	64	184	0	255
Value	83	255	0	50

After the background is removed the Canny edge detection is used on the black and white image of the head to produce an outline of the head. A minimum value of 60 and a maximum value of 100 is used for the Canny edge detection detailed in Section 2.6. The values were set when the Canny edge detector was operating on a grayscale image of the horse, but the grayscale image produced too many edges so it was abandoned for the black and white HSV filtered image. Once an outline of the horse's head is created the ear tip can be found.

To find the ear tip the center point of the top halter bounding box is used to initially, and to find the ear tip again if it lost during tracking. To initially find the ear tip the program looks

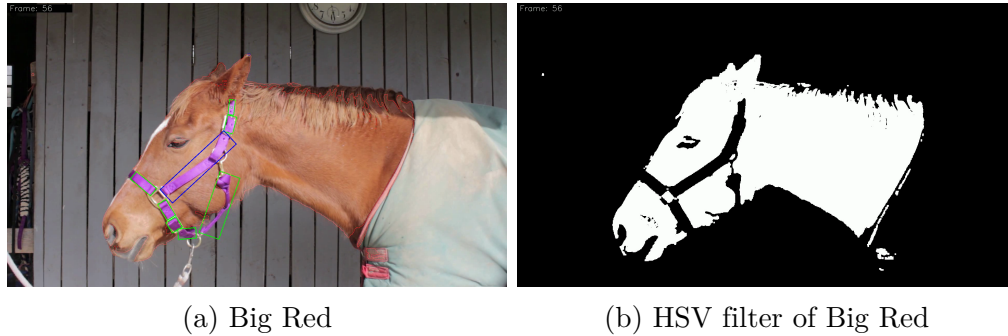


Figure 4.7: The original image of Big Red a light colored horse (left) and the HSV filter image (right).

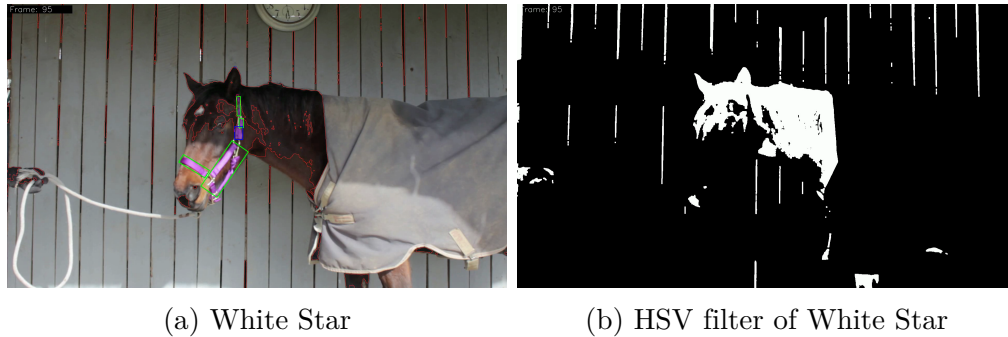


Figure 4.8: The original image of White Star a dark colored horse (left) and the HSV filter of White Star head (right). Shadows and the background made filtering the dark horses more difficult as can be seen in the left image as part of the head is missing and the background shows up in white.

for the highest place a Canny edge line crosses in the column above the top halter point. If it finds a line then the program moves left along that line as long as the line continues in a left or up direction. Once the line moves in a right or downward direction a flag is set to prevent the code from tracking back and forth infinitely. After the flag is set then the program follows the line to the right until the line moves left or downward. The point where there program ended is assumed to be the ear tip. If the there is no line in the column then the program moves left from the halter point until it finds a line. The program then looks for the highest point where a line crosses the column then follows that line left then right as described above. Figure 4.9 shows with blue arrows how the program starts at the top halter point in green, goes straight up then follows the red line to the ear tip.

After the initial ear tip is found the ear tip is tracked from the previous ear tip location. The code looks at a region of the Canny image 21 rows by 21 columns centered at the the previous location of the ear tip. The program goes to the top left most white pixel. From



Figure 4.9: The blue arrows show how the program starts at the top halter point in green, goes straight up then follows the red line to the ear tip.

that location the program tracks right and left using the same method describe above.

The background caused some issues with the dark horses, because there were gaps in between the boards of the barn, and behind the boards were two dark stalls. The gaps were a similar color to the dark horses. Looking at Figure 4.8b the gaps show up in white just like the head of a dark horse. Anytime the tip of the ear contacted a gap the ear tracker would move up the line and lose the ear tip thinking the gap is the ear. However, in future work could use distances from the halter to check and warn when a false reading is detected.

4.5 Pilot Work on Eye Detection

Eye tracking proved to be difficult as of the writing of this thesis eye tracking only works on light horses. Since the eye tracker looks for dark areas in the frame it can't distinguish between the dark eye and a dark horse. The eye tracker also does not work on horses with light eyes. There was only one case of a light eye available and the horse had to face to the right for that eye to be visible so that video was not used in the scope of this project. The eye tracker can mistakenly track shadows, or the horse's nostril.

The eye is invisible when the eye lid is shut making the eyes more difficult to continuously track. As the eye opens and closes it changes size and shape, which can provide information on the openness of the eye, makes the eye more difficult to track than the head or ear. Since the tracker has not yet been made as robust as the ear or head trackers there is no graph showing the status or the changes in location of the eye.

The eye tracker works similar to the halter tracker, looking for contours in a black and white image. Instead of using the contour function in OpenCV the SimpleBlobDetector was used. SimpleBlobDetector uses the contour functions but adds filtering parameters. The eye tracker uses the head HSV image. Two parameters are used to find the eye filterByColor, looking for black colored objects, The second parameter is filter by area looking for any blob made up of more than 250 pixels. The eye tracker returns the center point of the eye.

Figure 4.10 shows the eye what the eye tracker sees when Big Red has his eye closed. Big red has enough black around his eye that it could still be found when the eye is closed. Figure 4.10b shows that when the eye is closed two objects are now found by the eye tracker that confuse the eye tracker.

The eye is still an area that needs a lot of improvement, because the eye is a similar color to dark horses. Work is also needed to determine whether the eye is open or closed, this could be done by looking at the change of the area of the eye over time.

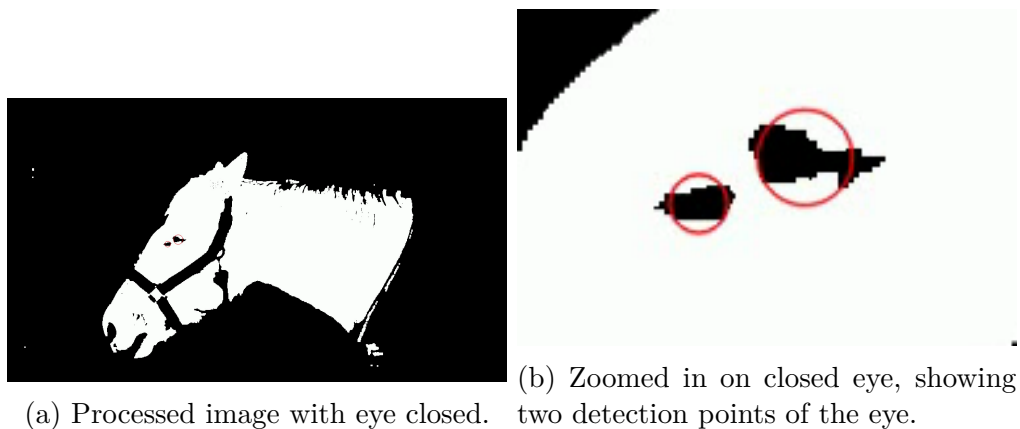


Figure 4.10: Eye closed changes the detection size, shape, and number of detections.

4.6 Data Presentation

The program does several things after it completely processes the video.

1. displays all of the graphs

2. saves all of the graphs as *.png format
3. saves all the data in comma separated value(csv) format
4. there is also a separate program that converts the csv back to python formatted graphs and *.png.

4.6.1 Motion Tracking Graphs

There are two lines per subplot the first is maroon that shows horizontal motion or motion along the x-axis. Figure 4.11 shows horizontal motion of a horses ear to the right and a simplified version of how the graph looks over time. When the ear moves to the right in the image the graph moves down and vise-versa for left. Figure 4.12 shows the the same thing for the head/halter motion in the horizontal direction.

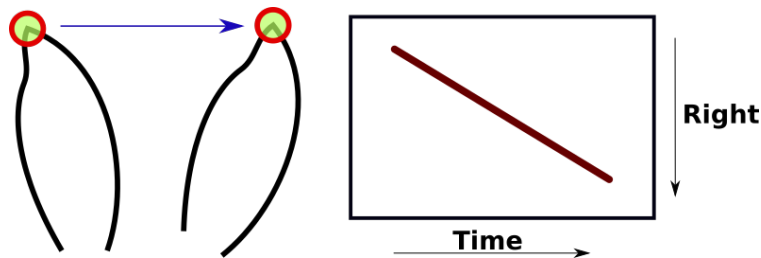


Figure 4.11: Right motion of the ear translates to downward motion of the maroon line on the ear graphs.

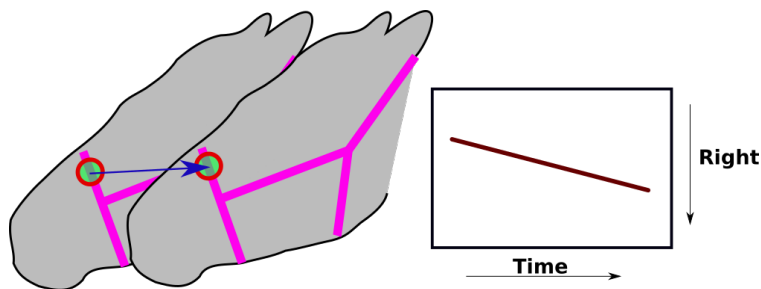


Figure 4.12: Just like the ear right motion of the head translates to downward motion of the maroon line on the halter graphs.

The second line is the orange line which is shown in Figure 4.13. The orange line shows vertical motion or motion along the y-axis. When the ear moves down the orange line also moves down. When the ear moves up the graph also moves up. Figure 4.14 shows the vectlcal motion for the head/halter.

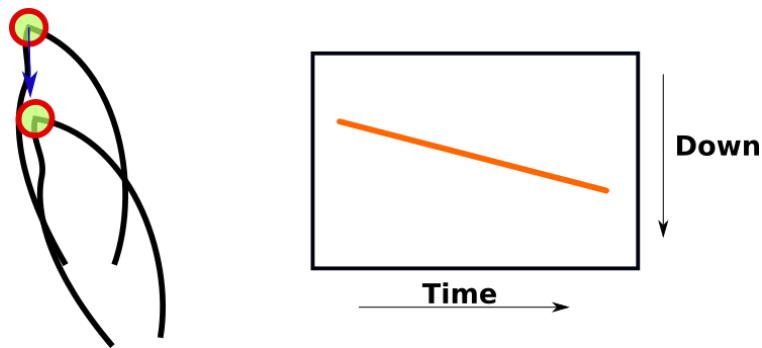


Figure 4.13: Down motion of the ear translates to a downward motion of the orange line on the ear graphs.

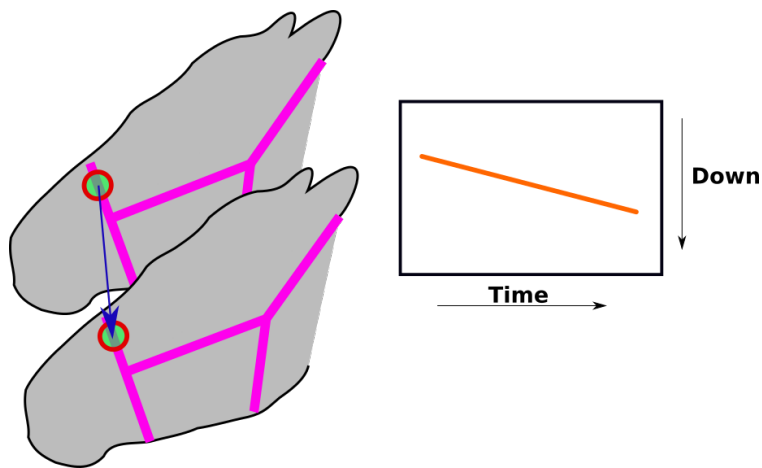


Figure 4.14: Just like the ear down motion of the head translates to downward motion of the maroon line on the halter graphs.

In the position subplot 0 is in the top left to match image in OpenCV where the 0,0 point is in the top left of the image. The units for position are pixels. The units for velocity and acceleration are pixels per millisecond (pixels/ms) and pixels per millisecond² (pixels/ms²)

Pixels can be translated to other units such as centimeters or inches, but require additional measurements taken at the time of videoing and camera calibration. Useful information is embedded in the pixel analysis, so real world distances were not pursued.

4.6.2 Velocity and Acceleration Tracking

The velocities and accelerations are graphed to see if they provide useful information. It is common to look at the change of position over a change in time, which gives you the velocity

also known as the first derivative (\dot{x}, \dot{y}) . Taking the derivative of the velocity with respect to time this gives you acceleration or the second derivative (\ddot{x}, \ddot{y}) acceleration is a useful to know because it allows you to determine the force on an object. Force equals mass times acceleration ($F = m \times a$) the mass is measurable and the acceleration can be determined from the change in position over time as mentioned above.

To find the first and second derivative the function shown below is used. The function takes two lists of the same length. The derivative function uses the `diff()` function from the numpy library. `diff()` takes the difference of each element in a list and outputs a new lists of the differences. The derivative function then uses the `divide()` function to divide each element in list a by each element in list b. Equation 4.1 shows mathematically what the `derivative()` function is doing, where a is element of the first list at index n , and b is the element of the second list at the same index as list a .

$$\frac{da_n}{db_n} = \frac{a_n - a_{n+1}}{b_n - b_{n+1}} \quad (4.1)$$

Listing 4.1: Code snippet from the main code of the derivative function.

```

1  import numpy as np
2
3  def derivative(a,b):
4      ''' takes the derivative of matrix a with respect to matrix b '''
5      da = np.diff(a, axis=0)
6      db = np.diff(b, axis=0)
7
8      return np.divide(da, db)

```

4.7 Accuracies

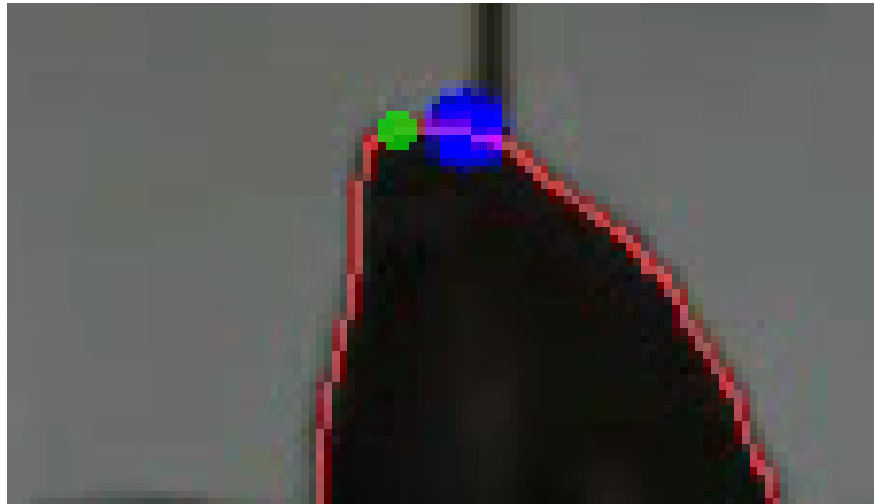
Three frames of seven different horses were looked at to determine accuracy of the ear tracker. For two of the seven horses the ear tracker did not track the ear. In the video of Tethys the ear farthest from the camera was tracked. The distance calculations were done based on if that was the preferred ear, because the tracker did correctly track an ear. From 15 frames the minimum distance was zero pixels, the max was ten pixels and the average was four pixels. Table 4.3 shows all of the distances calculated from each frame.

Table 4.3: All of the distances between what the computer determined and the author determined as the ear tip, in pixels, for three random video frames from seven horses. Overall average distance is 4.7 pixels

Horse	Frame 1	Frame 2	Frame 3	Average
Big Red	4	3	0	2.3
White Star	1	8	3	4
Chief	10	2	4	5.3
Mosfet	No ear track			
Shaggy Pony	9	8	6	7.7
Titanic	No ear track			
Tethys	4	4	4	4



(a) Full image of White Star.



(b) Zoomed in image of ear to show eight pixel distance between author(green) and computer(blue) ear tip locations.

Figure 4.15: Full image of White Star(a), and a zoomed in image of the ear(b) to show the eight pixel distance between authors ear tip location(green) and the computers ear tip location(blue). The diameter of the green dot is 5 pixels and ten pixels for the blue.

Chapter 5

Results

This chapter details results of application of automatic head and ear detection and tracking in videos of 9 horses, comprising 7 minutes of video footage. Additional footage of 2 horses were taken and not used because the handler was in the frame wearing a similar color to the halter, and one horse was uncooperative when facing to the left. The program only works on horses that are facing left.

5.1 Data Presentation

This section explains how data from the auto tracking algorithm is presented, this includes head, and ear tracking.

5.1.1 Head tracking

This section shows the results of the head tracker discussed in Section 4.3. First, two examples are shown to explain the type of data collected and how it is represented graphically. For example, data from videos of a pony, The Dude, shown in Figure 5.1 are used. In this scenario The Dude looks at the camera twice during filming. The second example is a video of Big Red that shows what happens when a horse looks past the camera shown in Figure 5.5.

Recall Figure 4.12 that shows when the horse's head moves to the right the maroon line moves down. Figure 4.14 shows when the head moves down the orange line in the graph moves down.

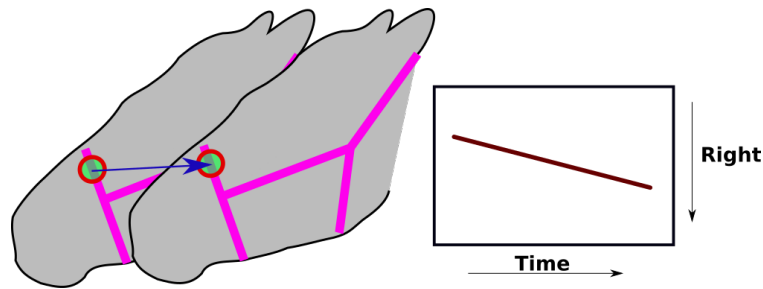


Figure 4.12: Right motion of the head translates to downward motion of the maroon line on the halter graphs. (repeated from page 32)

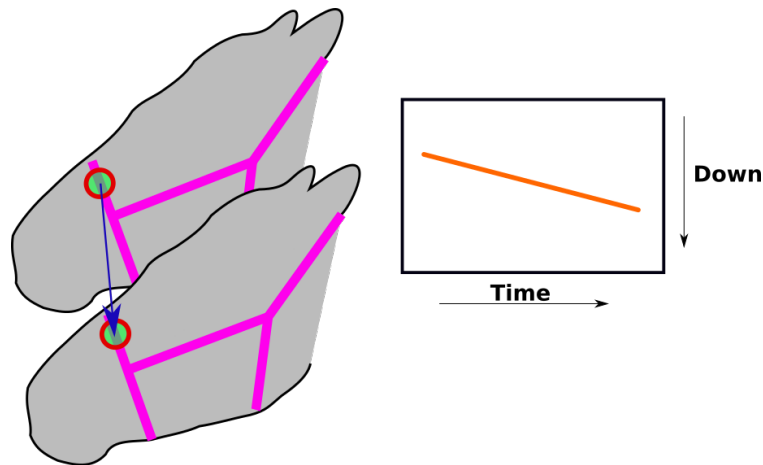


Figure 4.14: Down motion of the head translates to downward motion of the maroon line on the halter graphs. (repeated from page 33)

Figure 5.2 shows the head motion graph from a clip of *The Dude*. In the clip *The Dude* looks at the camera twice and the head tracker does not lose its track. The two dips in the maroon line are when *The Dude* turns his head to look at the camera. Then after that his head slowly moves down starting at time $t = 20,000$ ms as he starts to relax. The motion can be clearly seen in the position and velocity graphs. The acceleration is not greater than the noise in the graph so it does not show any discernible information. The motion of how the head moves can be more easily seen in the 2-Dimensional purple to yellow plot of Figure 5.3. Each point on the plot represents the same point on the horse's head as it moves throughout the video frames and the color represents the time. The blue to green to yellow plot shows the ear motion which is discussed in the next section.



Figure 5.1: Photo of The Dude

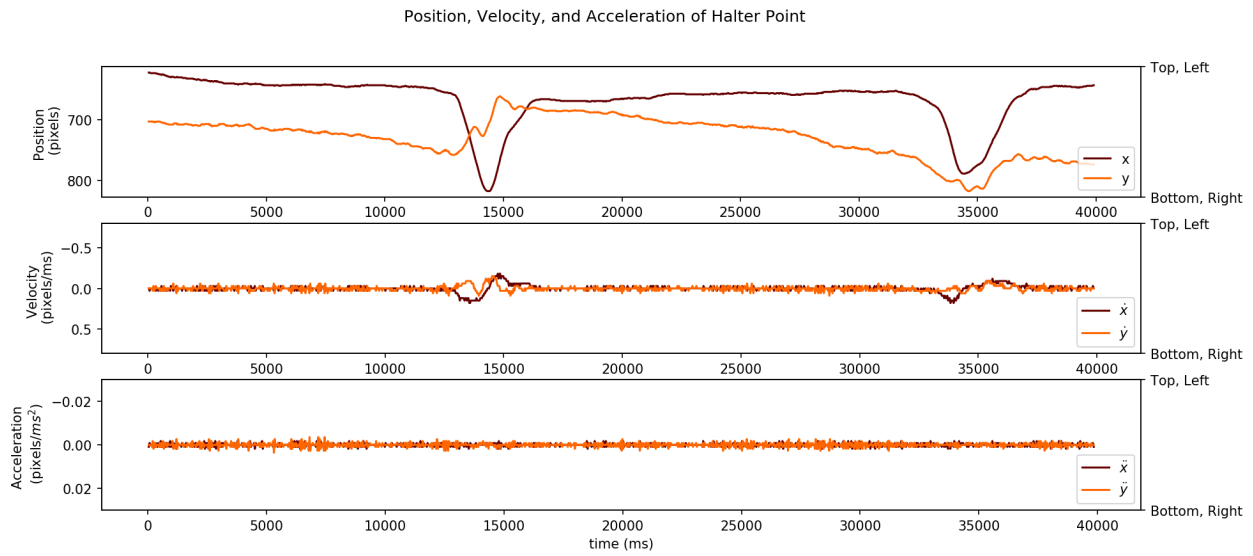


Figure 5.2: Head motion graph of The Dude.

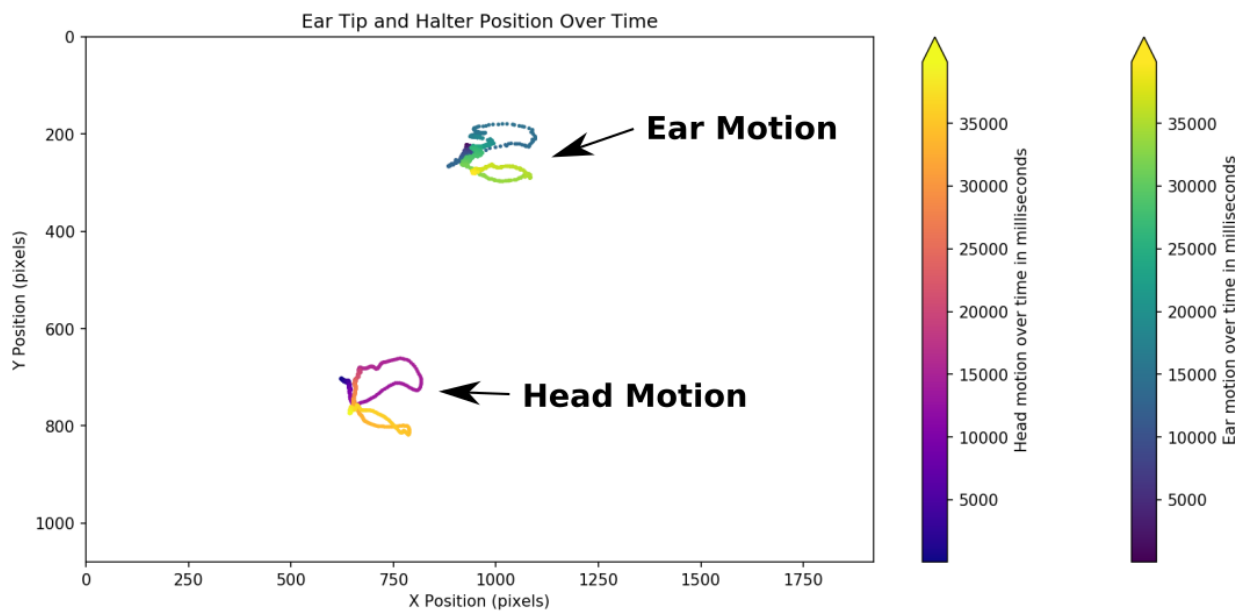


Figure 5.3: 2-Dimensional Head motion graph of The Dude.

Another example this one of Big Red as shown in photos of Figures 5.4 - 5.6 the data here shows what happens when a horse looks past the camera and the other side of the halter becomes visible. Horses are curious animals so when a new object and person are in their normal environment, the horses want to look at it, When a horse looks past the camera the other side of the halter becomes visible as shown in Figure 5.5 causing jumps in the graph, and this is shown as point B in Figure 5.7. The graph in Figure 5.7 shows results of the head tracker for the first ten seconds of the clip 4893_edit00_1920_1080.mov. The graph is Labeled with A, B, and C that correspond to the A in Figure 5.4, B in Figure 5.5, and C in Figure 5.6. Figure 5.4 shows frame zero of the clip where Big Red is looking forward shortly after that Big Red turns to look at the camera and looks a little past the camera. The other side of the halter comes into view around frame 61, shown in Figure 5.5 and B on the graph, this is where the graph jumps because head tracker tracks the left most purple object which which is normally the nosepiece, becomes the right cheekpiece. Big Red then turns his head back to look forward shown in Figure 5.6. The head tracker briefly tracks a different object, but returns to tracking the head when Big Red looks forward, and for the rest of the video the head tracker tracks the same point. The large peaks in the acceleration and velocity could be used to detect jumps in the tracking, and then compare the position before and after the spike to determine if a different point is being tracked this was left for future work.



Figure 5.4: The first frame of the video to show the initial head position.



Figure 5.5: Frame 61 where Big Red looks past the camera and the head tracker tracks the opposite side of the halter that is denoted by the left most green box with a blue dot at the center that might be difficult to see.



Figure 5.6: A frame shortly after Big Red looks forward again after looking at camera. Frame 121 denoted in the top graph of Figure 5.7.

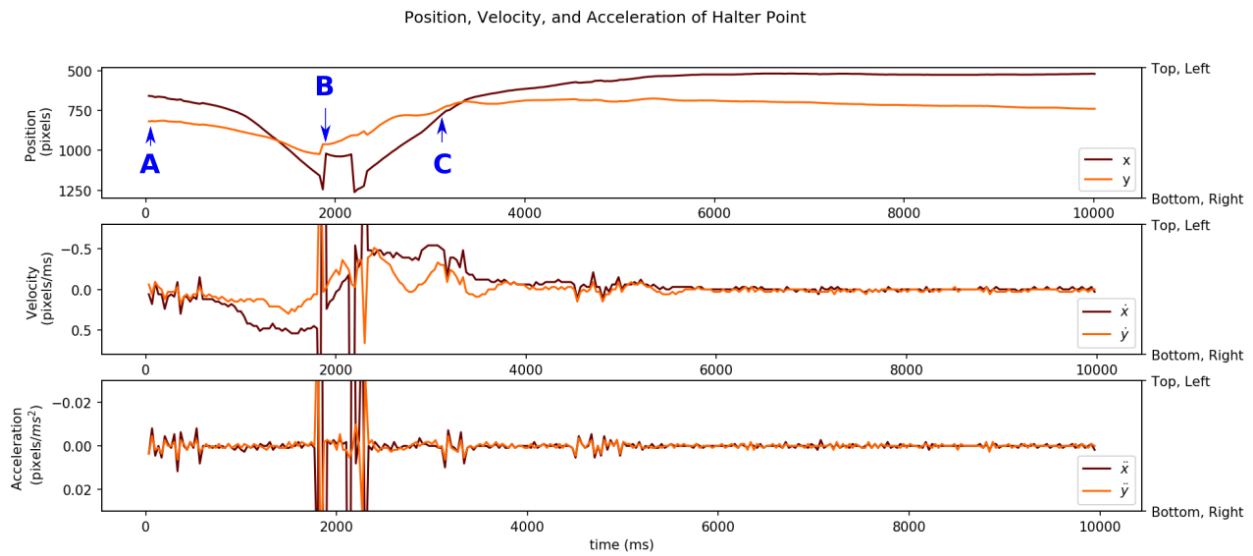


Figure 5.7: The first ten seconds of clip 4893 edit 00. The graph shows when Big Red looks to the left past the camera. A, B, C correspond to Figures 5.4, 5.5, 5.6.

5.1.2 Ear Tracking

All ear position data is given relative to the camera frame and not the head position, because of this any head motion also shows up in the ear motion. The motion of the head can be subtracted from the ear motion relatively easily in future work. The calculations required for removing the head motion from the ear motion was not done because it is beyond the scope of this project.

This section shows the results of the ear tracker described in Section 4.4. The ear tracking worked best on light horses, but also worked well on dark but had a few issues that were created by the background being a similar color as the horse. Turnout blankets on the horses that were similar color to the halter also effected the results.

Recall that when the ear of a horse moves to the right the maroon line moves down as shown in Figure 4.11. When the ear moves down the orange line in the graph moves down shown in Figure 4.13.

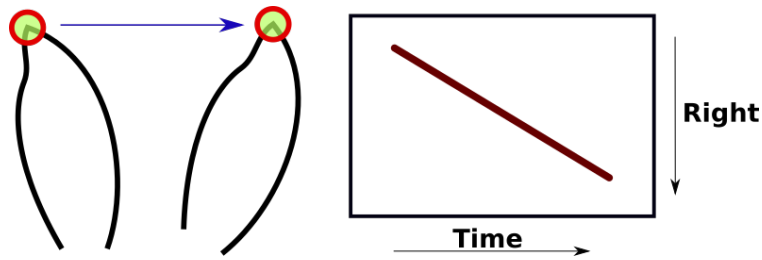


Figure 4.11: Right motion of the ear translates to downward motion of the maroon line on the ear graphs. (repeated from page 32)

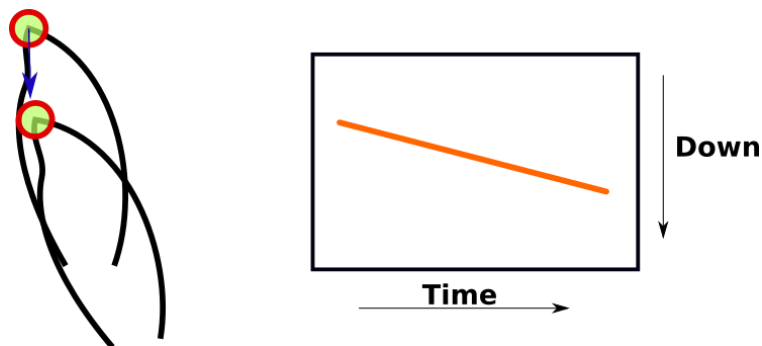


Figure 4.13: Down motion of the ear translates to downward motion of the maroon line on the ear graphs. (repeated from page 33)

The best example of ear motion is of Shaggy Pony pictured in Figure 5.8 in the clip 4905 edit 00. There is very little head motion in this clip as can be seen from the purple-red-yellow plot in Figure 5.10 as all of the points are in a single small grouping. The purple-green-yellow graph moves in a line as the ear moves forward and backward.



Figure 5.8: Photo of Shaggy Pony

Another good example of ear motion for a different horse is White Star's ear movement represented in Figure 5.11. This clip has some head motion, but the ear motion can be seen separate from the head motion. In this clip White Star slowly turns his head to the camera then turns away from the camera to walk forward. Recall left movement in the frame is forward movement to the horse. While White Star moves his head around, his ear moves backwards and forwards four times.

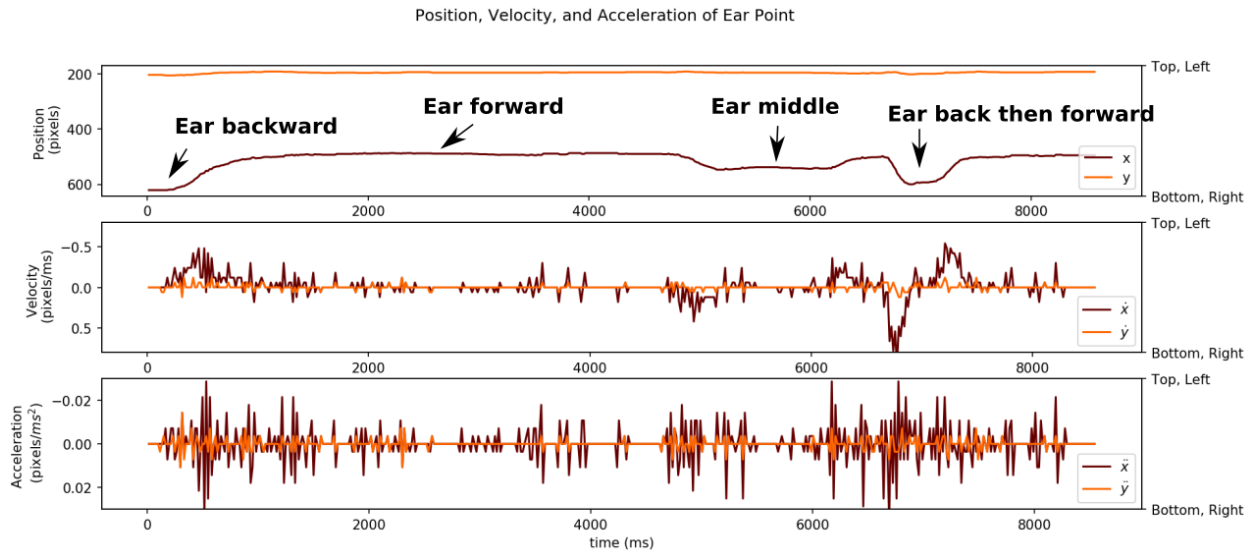


Figure 5.9: Ear motion of Shaggy Pony

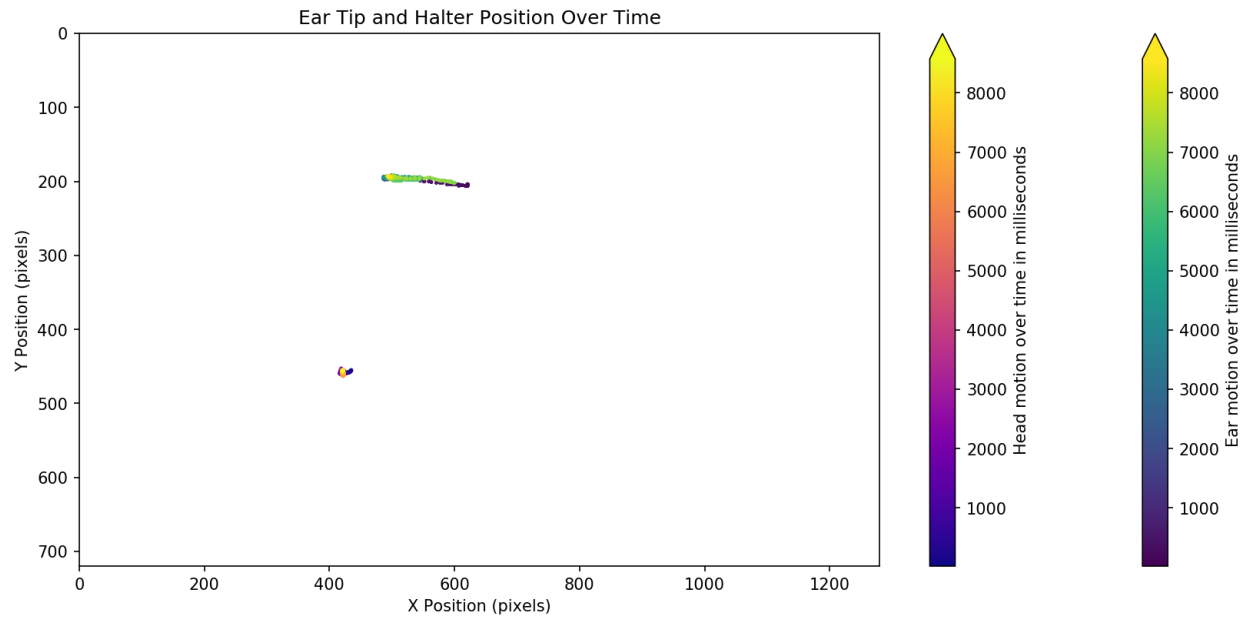


Figure 5.10: 2D graph of head and ear motion of Shaggy Pony

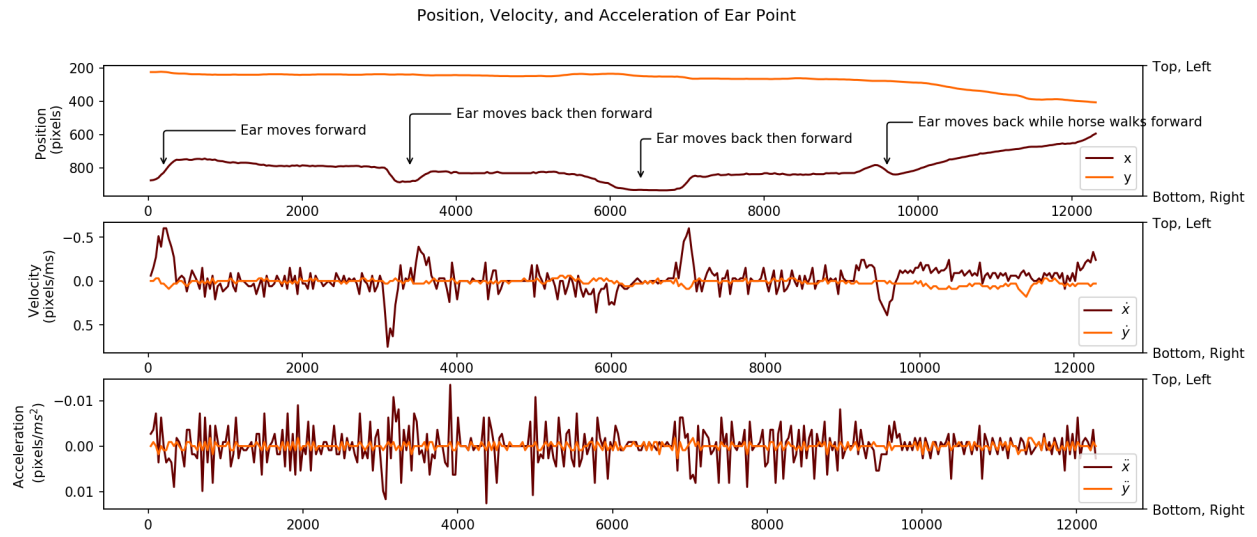


Figure 5.11: Ear motion of White Star



(a) Ear forward

(b) Ear back

Figure 5.12: Photo of White Star with ear forward and ear back.

5.1.3 Filtered Data

Two examples from above have been used to show when the data is filtered. The data in this section has been filtered so the noise does not get multiplied in the velocity and acceleration plots. All motion greater than 0.5 Hz has been filtered out. The position data has also been shifted to start at 0 to aid in filtering. The first example shown is when Big Red looks past the camera and the second example is of Shaggy Pony where there is ear motion but very little head motion.

Head and Ear Motion on Big Red

The following graphs are of the clip where Big Red looks past the camera. The filtering caused the position to start at (0,0) and caused

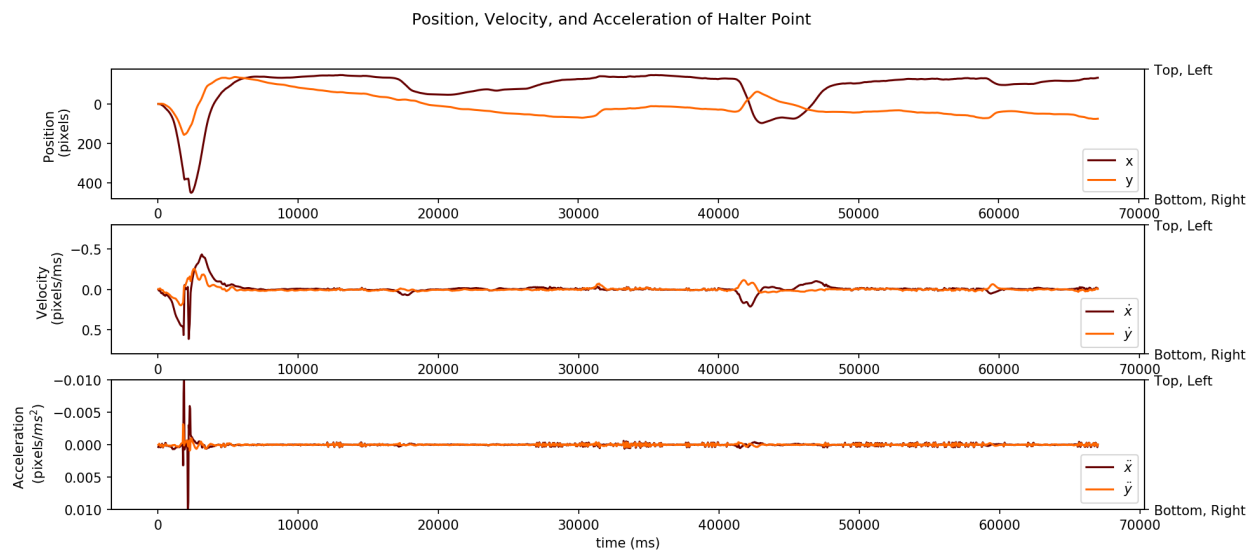


Figure 5.13: Filtered head motion of Big Red.

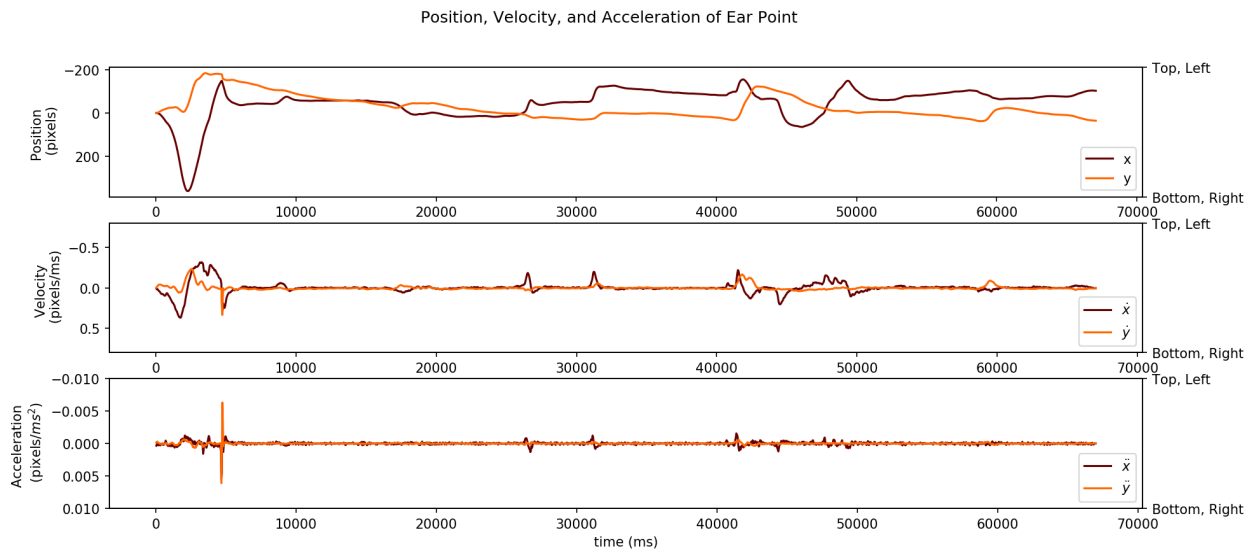


Figure 5.14: Filtered ear motion of Big Red.

Ear Motion Only on Shaggy Pony

The following graphs are of the clip of Shaggy Pony with just ear motion and very small head motion. The filtering caused the position to start at (0,0) and caused

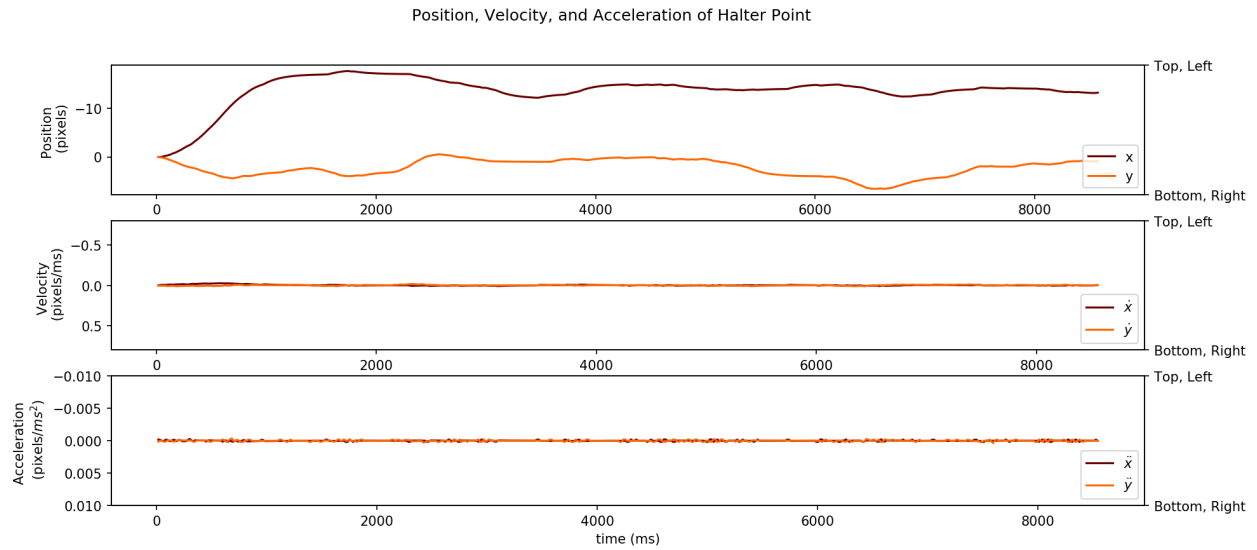


Figure 5.15: Filtered head motion of Shaggy Pony.

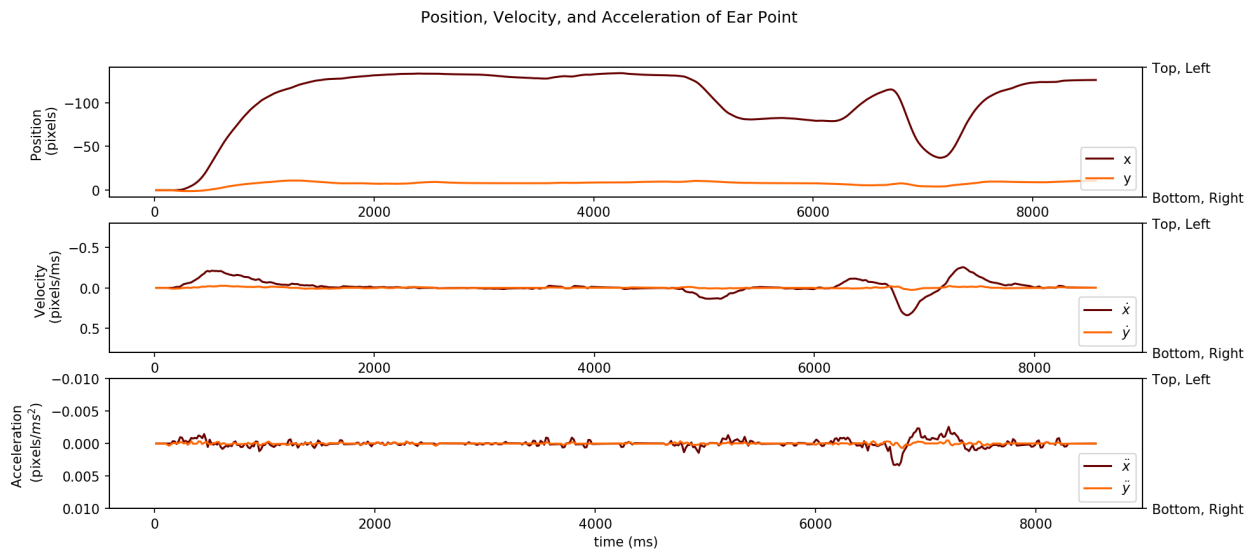


Figure 5.16: Filtered ear motion of Shaggy Pony.

5.2 Head Tracking and Ear Tracking Results

This section shows head tracking data and ear tracking data in one ten second clip for each horse. In some cases only shorter clips were available because of handlers in frame, horse walking out of frame, or other items that may confuse the computer program. If there is not a clip for a horse that is ten seconds long, then empty space was added to the end of the graph to make all graphs have the same time scale. The ten second clip was not always taken from the beginning of the video therefore the graphs do not always start at zero. Instead they start at the same time that the ten second clip started in the original clip, that way it is easier to compare the ten second clips in this section to the full length clip graph in Appendix B. For example if the ten second clip is seconds 5 to 15 in a 20 second clip then the graph of the ten second clip will start at 5 seconds or 5000 milliseconds. The position graph has a different y scale for each graphs to show subtle movements, but all the other scales are the same to make it easy to compare.

5.2.1 Big Red

This is a different clip than the one shown of Big Red in Section 5.1.1. In this clip Beg Red's ear moves forward and he turns to look at the camera the ear moves back then he turns to look froward.



Figure 5.17: Image of Big Red.

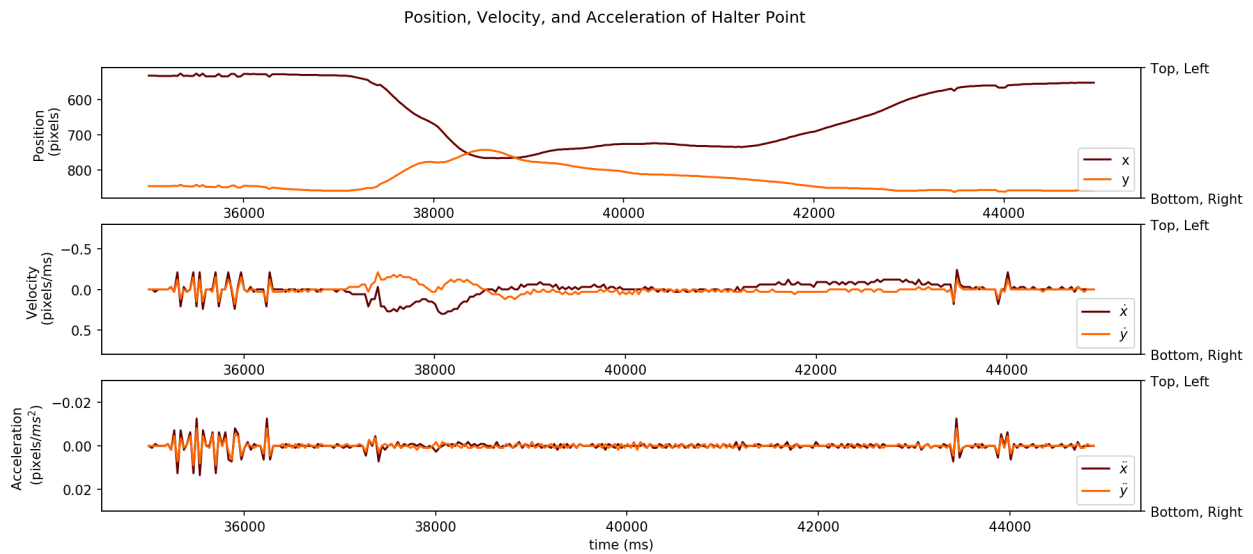


Figure 5.18: The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).

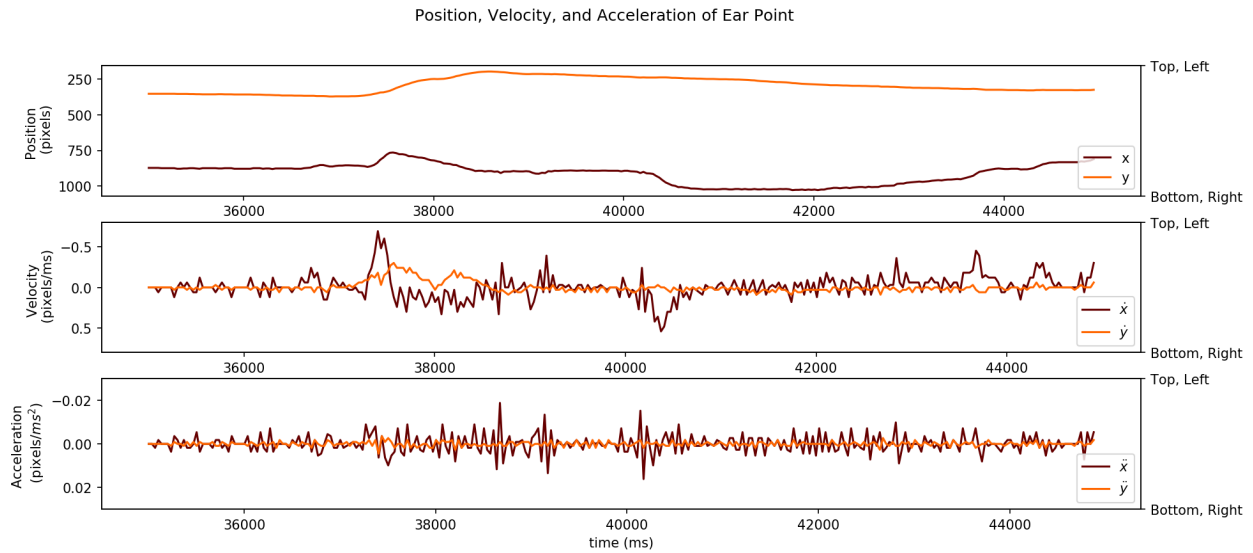


Figure 5.19: Orange is vertical motion, and maroon is horizontal motion of the ear tip.

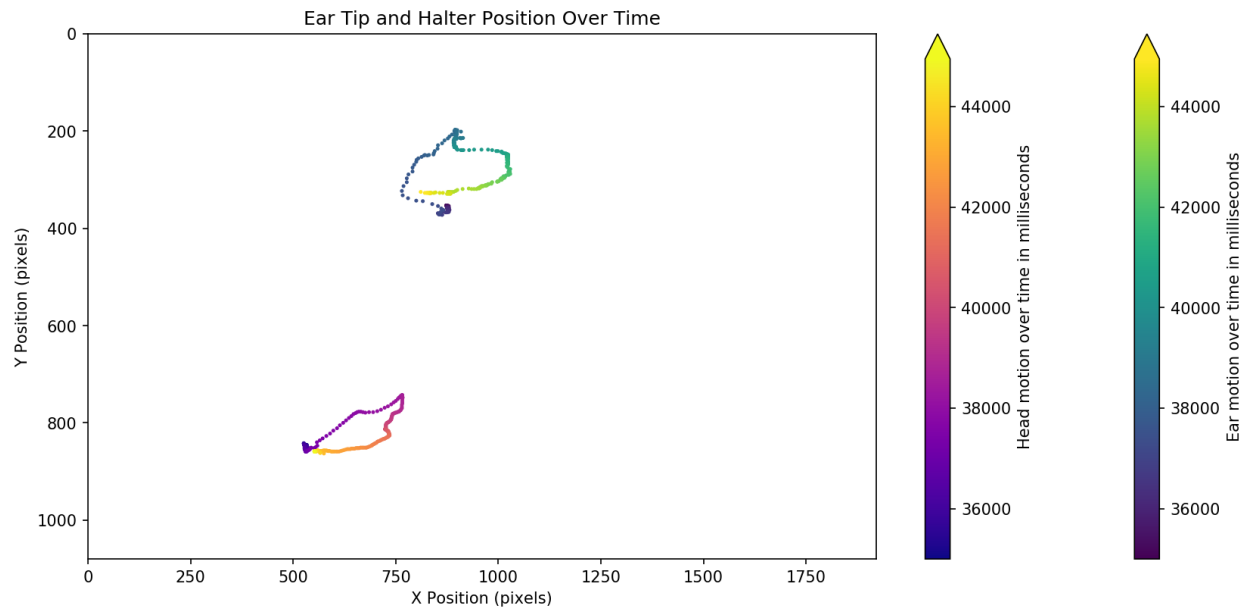


Figure 5.20: 2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicated by purple to green to yellow, and head motion is indicated by purple to red to yellow.

5.2.2 White Star

In this clip White Star's ear moves back then forward three times, seen as the three dips in Figure 5.23. Then at the end of the clip he starts walking forward that can be seen as the maroon line moves up in Figure 5.23, but more definitively in the maroon line at the end of Figure 5.22.



Figure 5.21: Image of White Star.

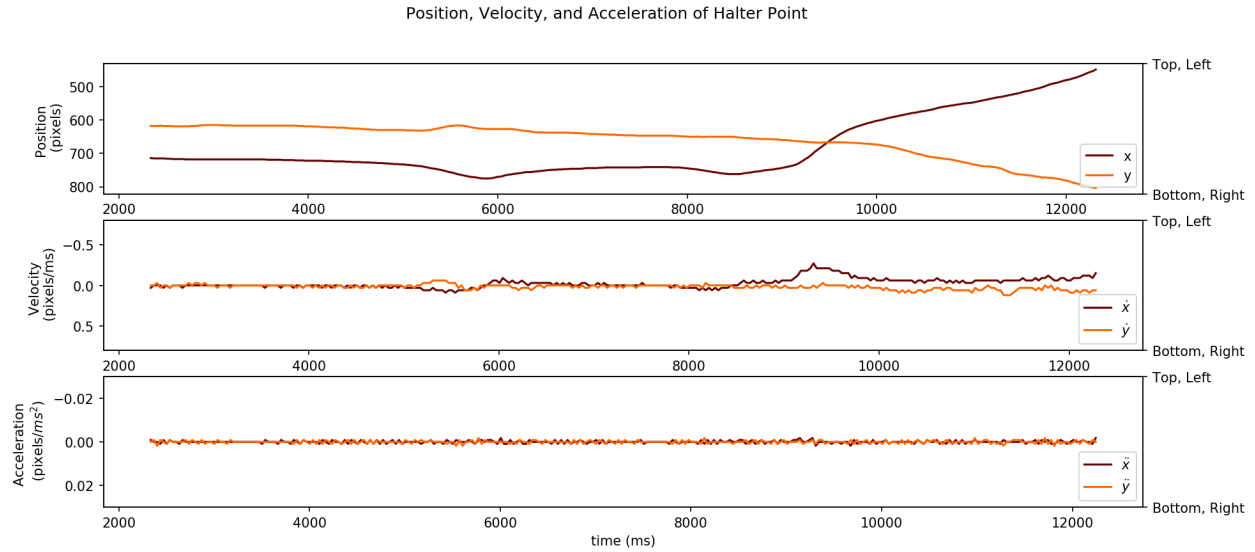


Figure 5.22: The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).

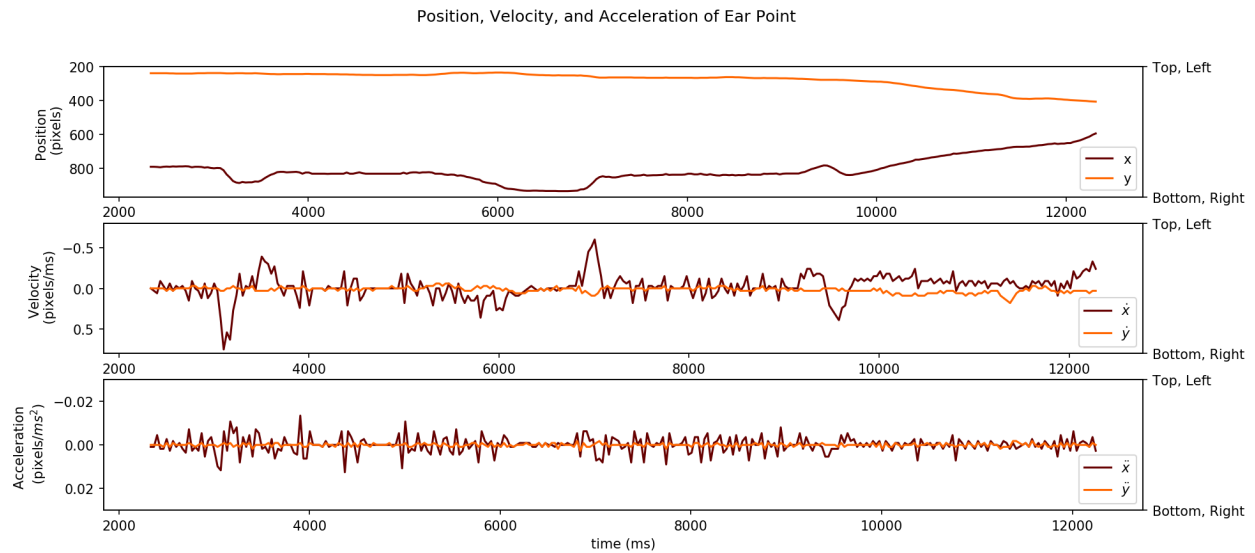


Figure 5.23: Orange is vertical motion, and maroon is horizontal motion of the ear tip.

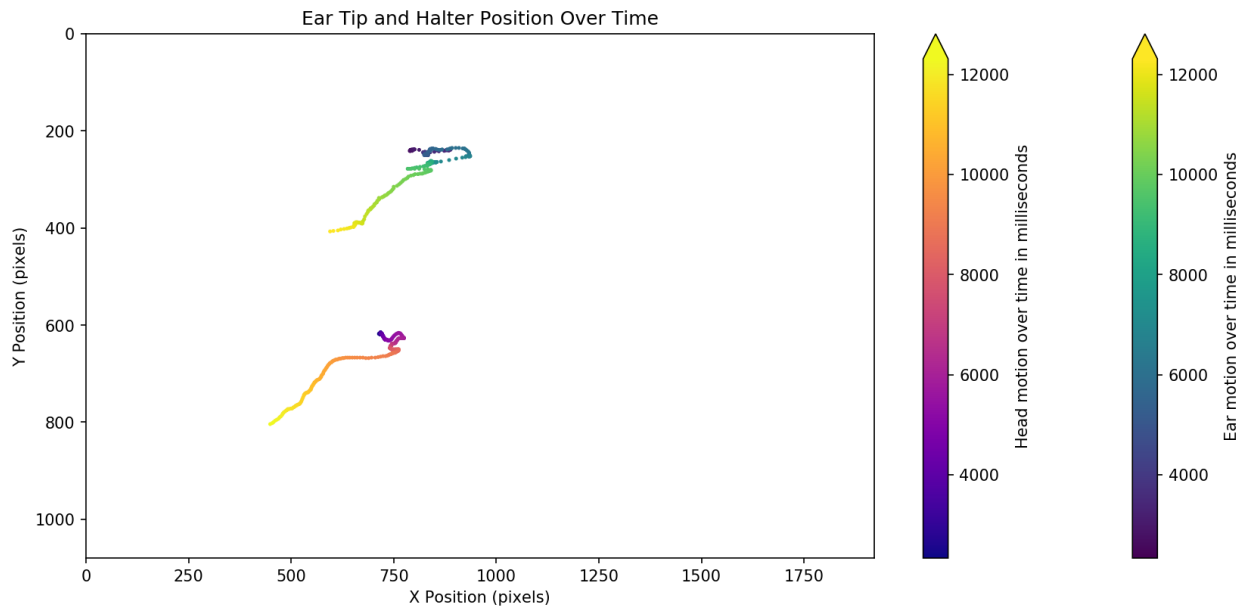


Figure 5.24: 2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicated by purple to green to yellow, and head motion is indicated by purple to red to yellow.

5.2.3 Big Benny

In this clip Big Benny is looking away from the camera and turns to look further from the camera. The head tracker starts off tracking the cheekpiece then jumps back and forth to the crownpiece. Then as Big Benny turns to look at the camera the head tracker tracks the nose piece. The ear tracker does not start tracking the ear until the last spike in the graph.



Figure 5.25: Image of Big Benny.

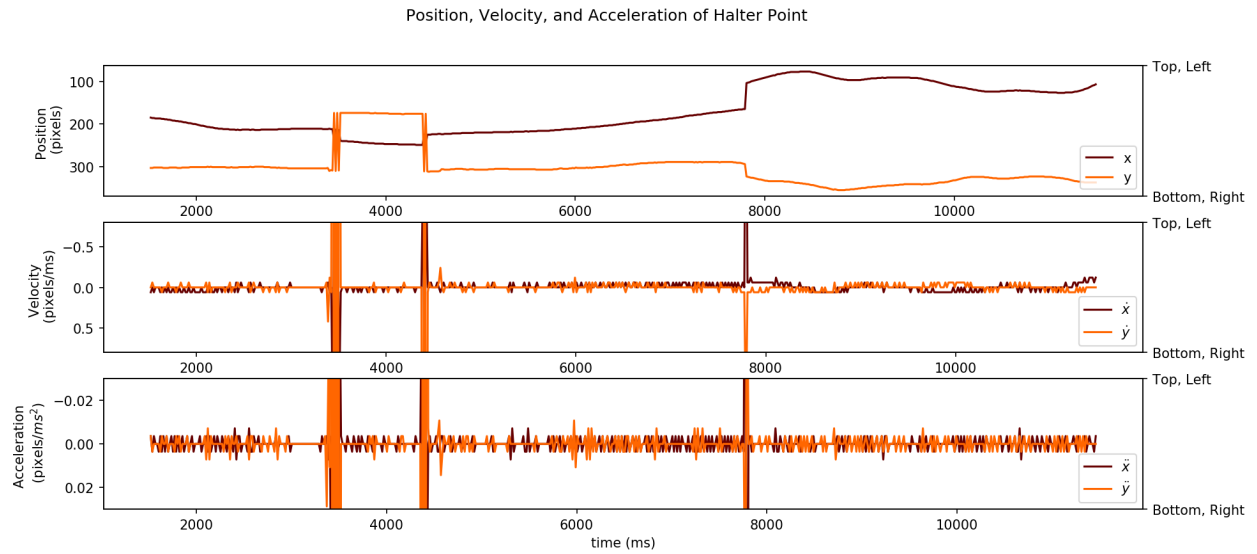


Figure 5.26: The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).

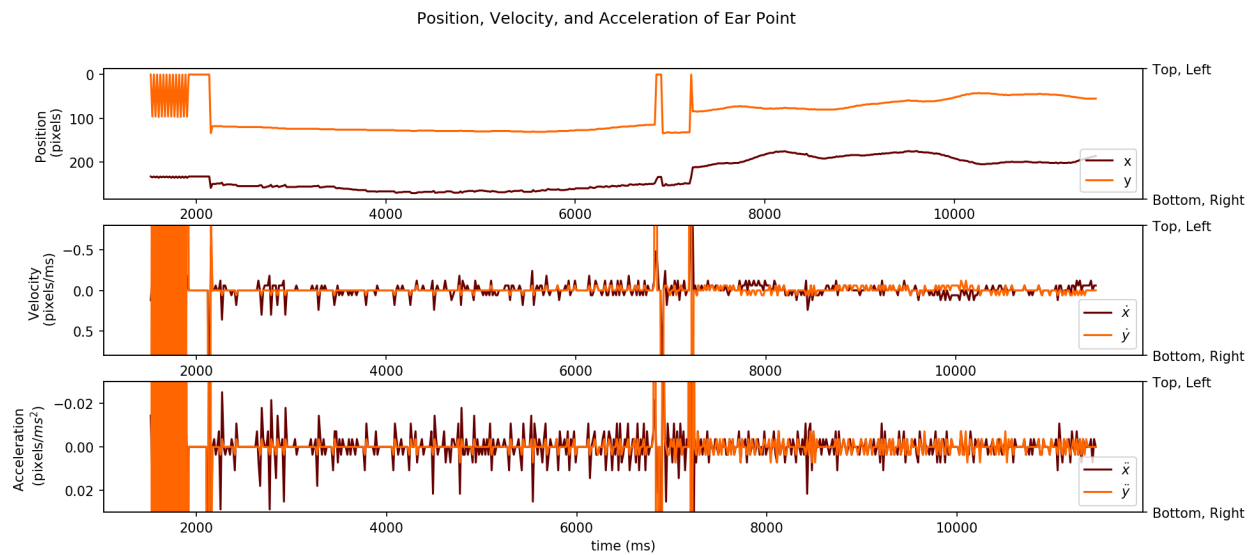


Figure 5.27: Orange is vertical motion, and maroon is horizontal motion of the ear tip.

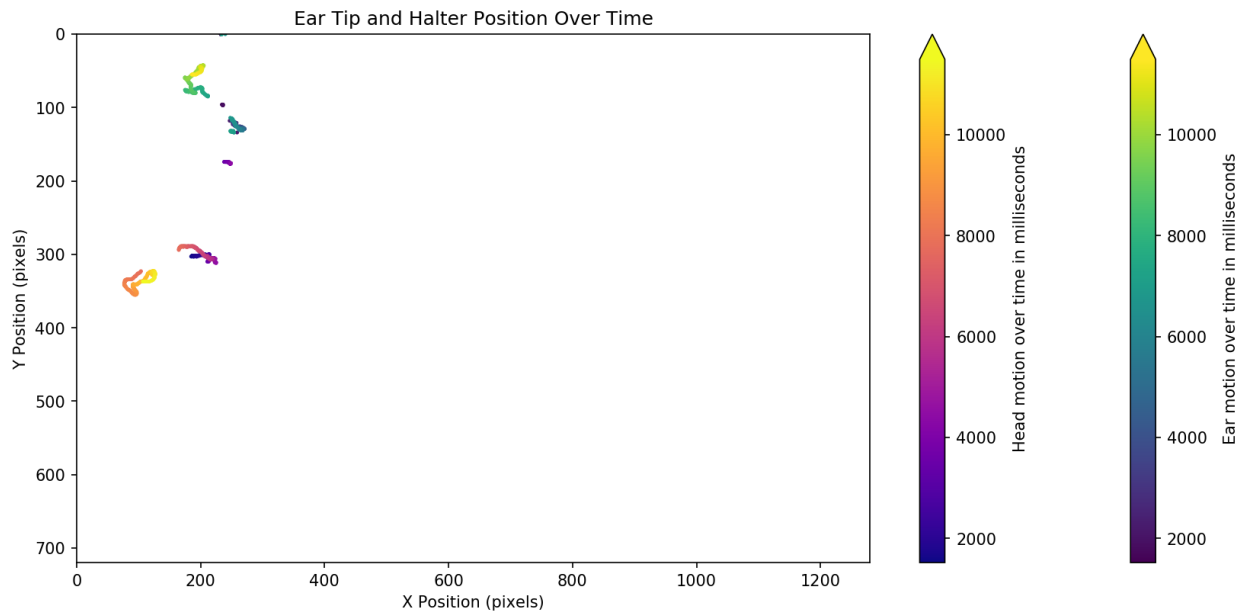


Figure 5.28: 2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicated by purple to green to yellow, and head motion is indicated by purple to red to yellow.



Figure 5.29: Image of Big Benny to show the head tracker tracking the chinpiece shown as the blue dot. Also notice the green boxes on the back that show the program thinks the blanket is part of the halter.



Figure 5.30: Another image from the same clip that show the head tracker now tracking the cheekpiece. The tracking of different parts of the halter show up as near vertical lines on position, velocity, acceleration graphs.

5.2.4 Chief

This clip starts with Chief looking at the camera and then looks away at the end of the clip.



Figure 5.31: Image of Chief.

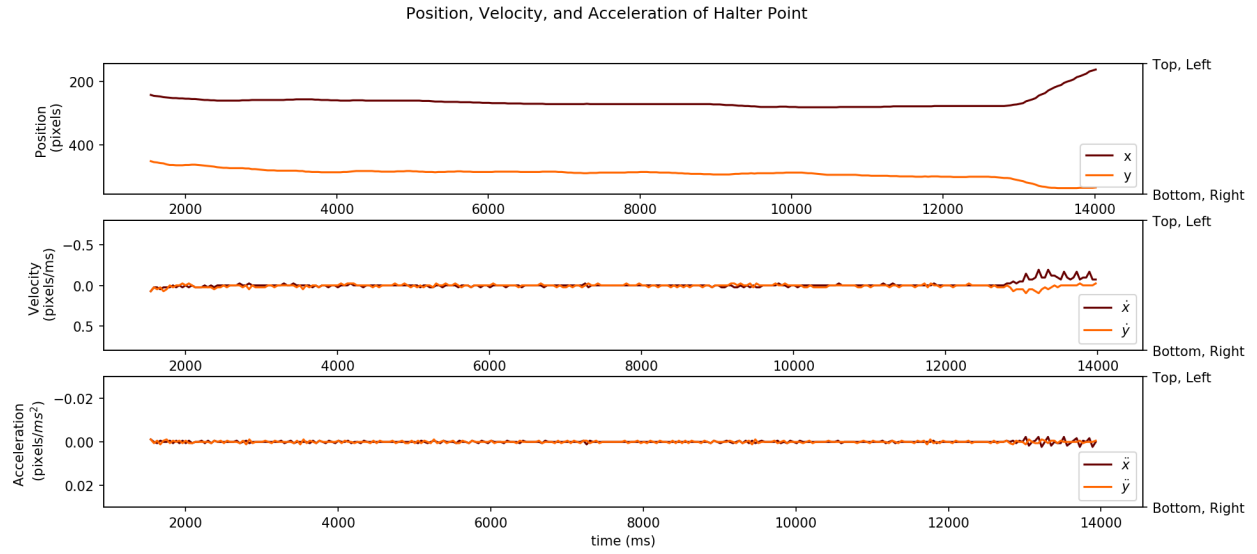


Figure 5.32: The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).

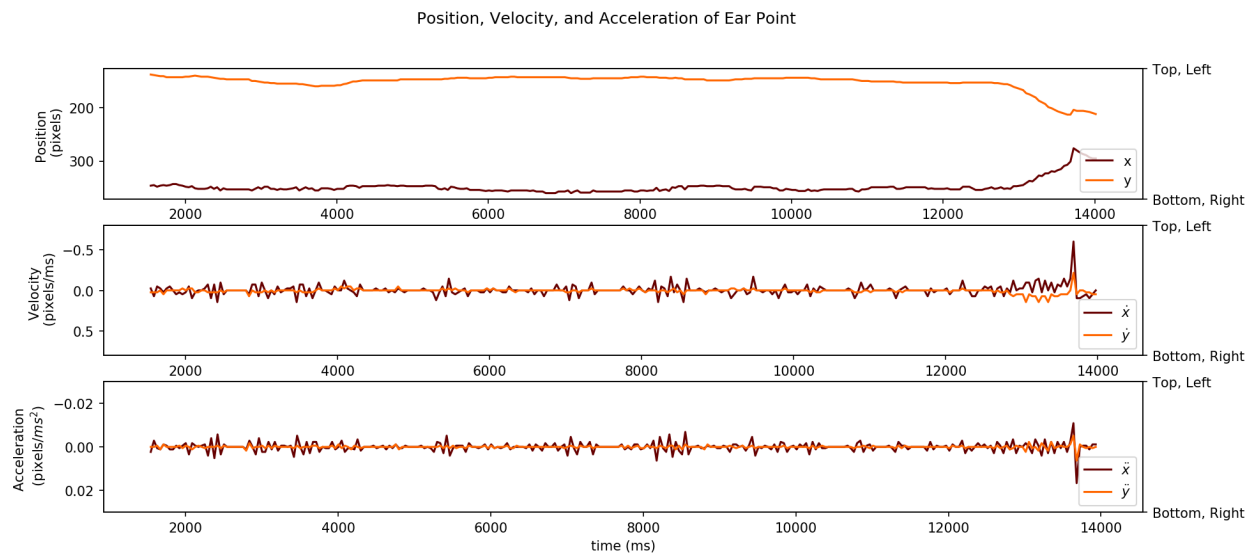


Figure 5.33: Orange is vertical motion, and maroon is horizontal motion of the ear tip.

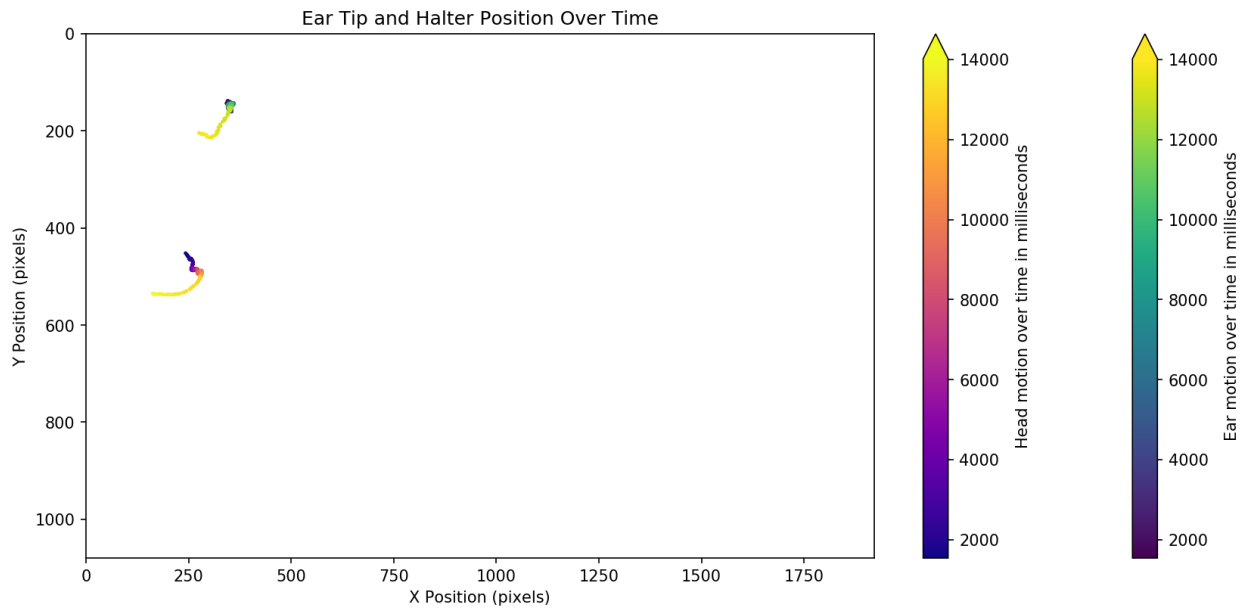


Figure 5.34: 2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicated by purple to green to yellow, and head motion is indicated by purple to red to yellow.

5.2.5 Mosfet

In this clip Mosfet's ear moves backward and his head subtly turns forward.



Figure 5.35: Image of Mosfet.

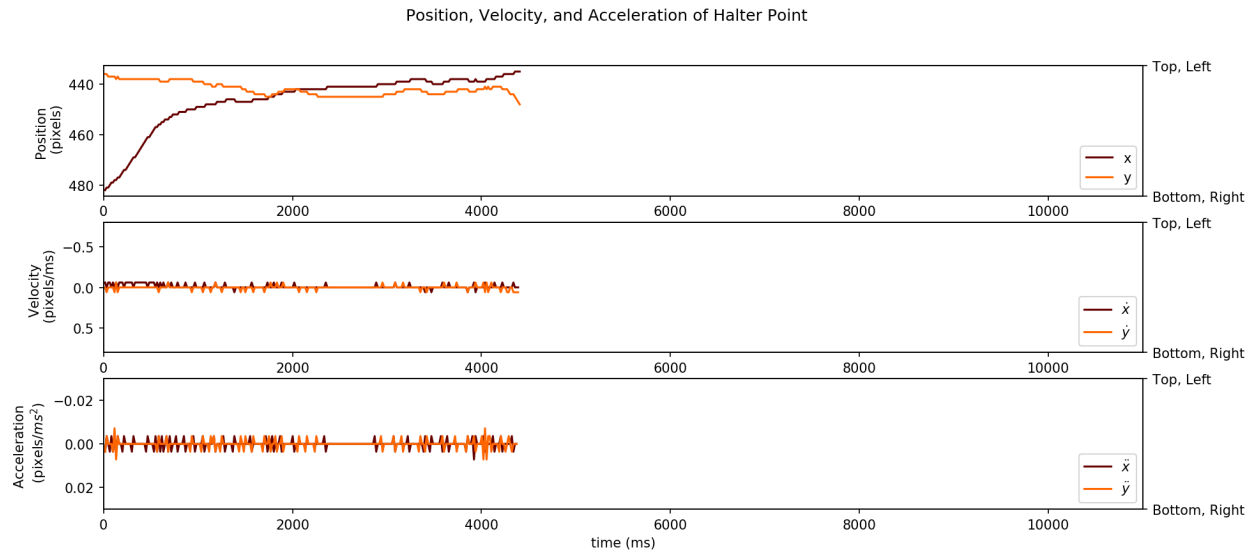


Figure 5.36: The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).

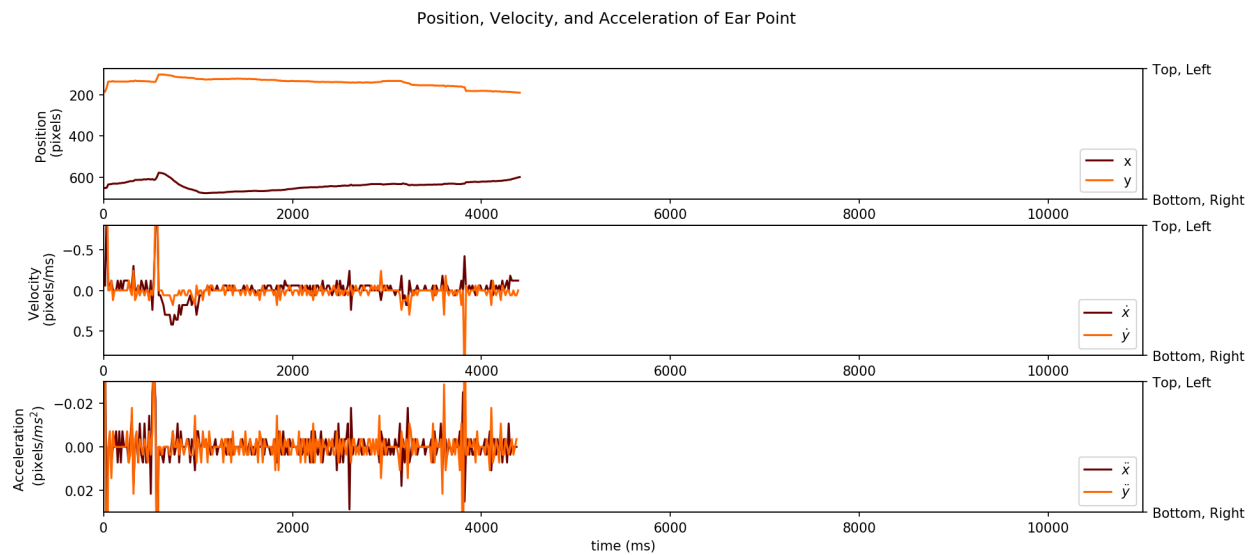


Figure 5.37: Orange is vertical motion, and maroon is horizontal motion of the ear tip.

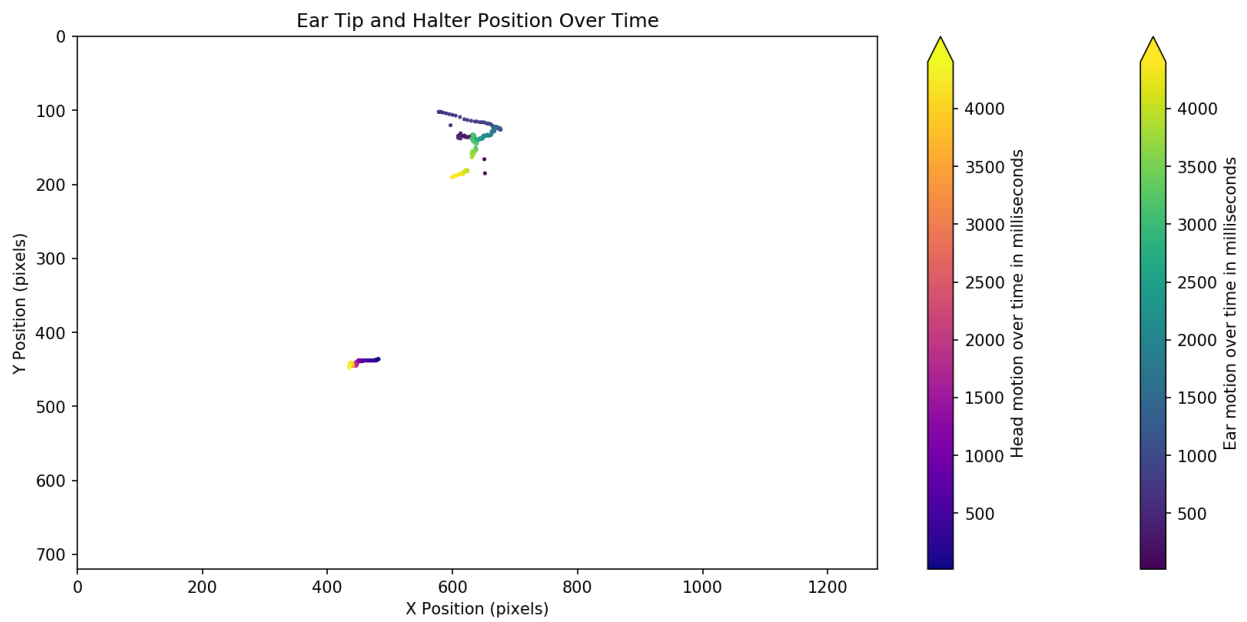


Figure 5.38: 2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicated by purple to green to yellow, and head motion is indicated by purple to red to yellow.

5.2.6 Shaggy Pony

In this clip Shaggy Pony moves his ear forward then to the side, then forward again. Shaggy pony then moves his ear back then forward.



Figure 5.39: Image of Shaggy Pony.



Figure 5.40: The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).

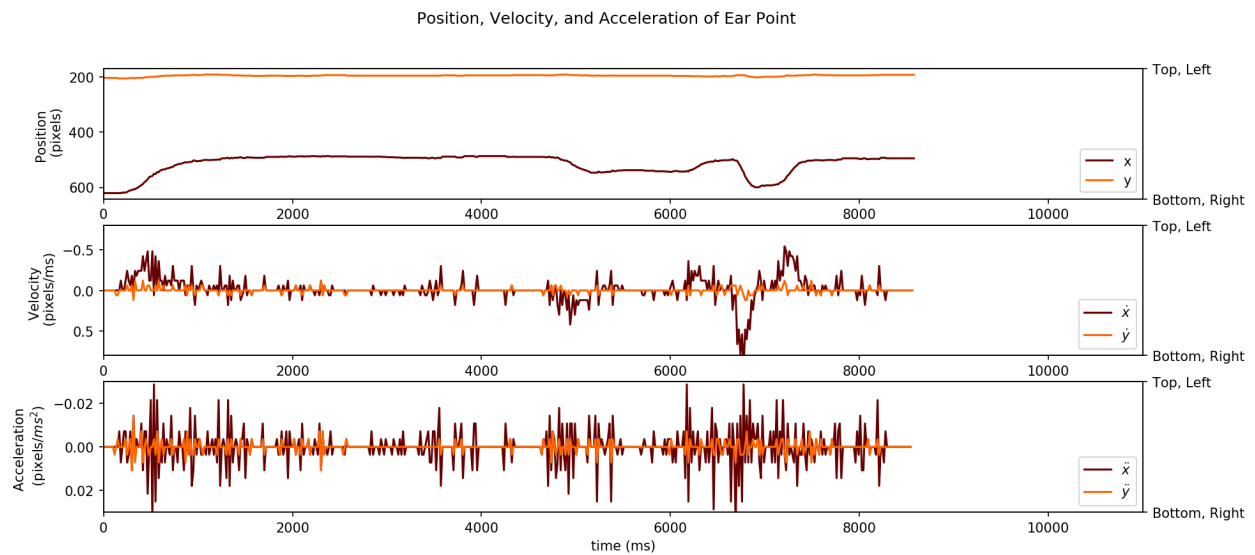


Figure 5.41: Orange is vertical motion, and maroon is horizontal motion of the ear tip.

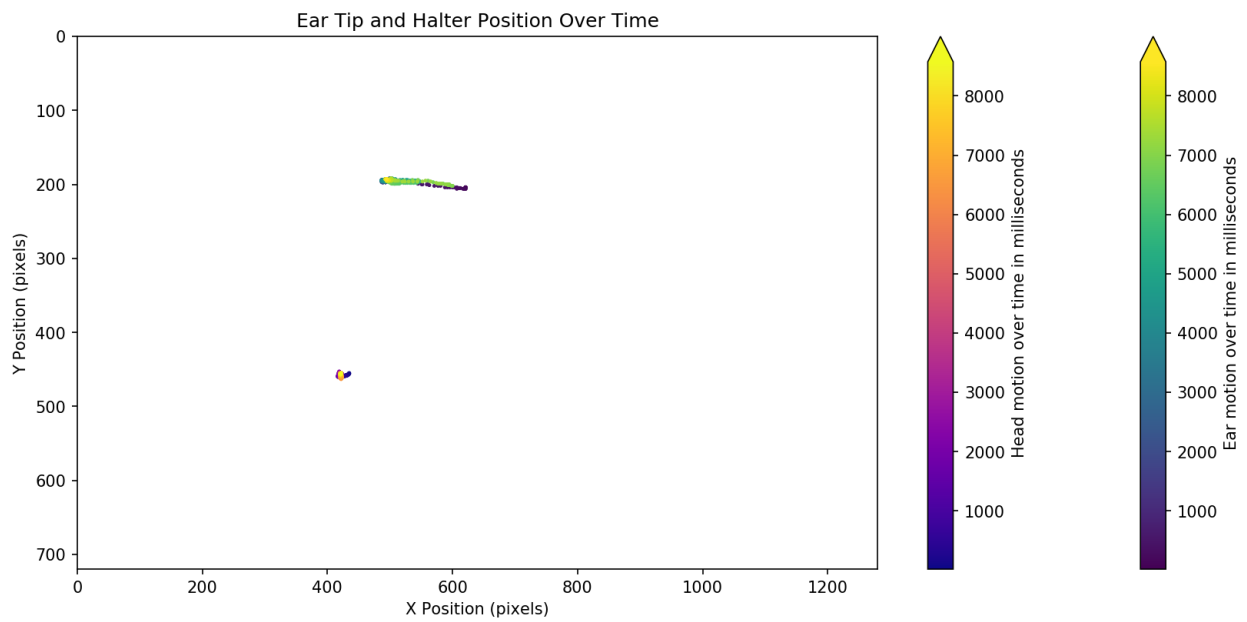


Figure 5.42: 2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicated by purple to green to yellow, and head motion is indicated by purple to red to yellow.

5.2.7 The Dude

The Dude's head turns to look at the camera then back forward with subtle ear movement.



Figure 5.43: Image of The Dude.

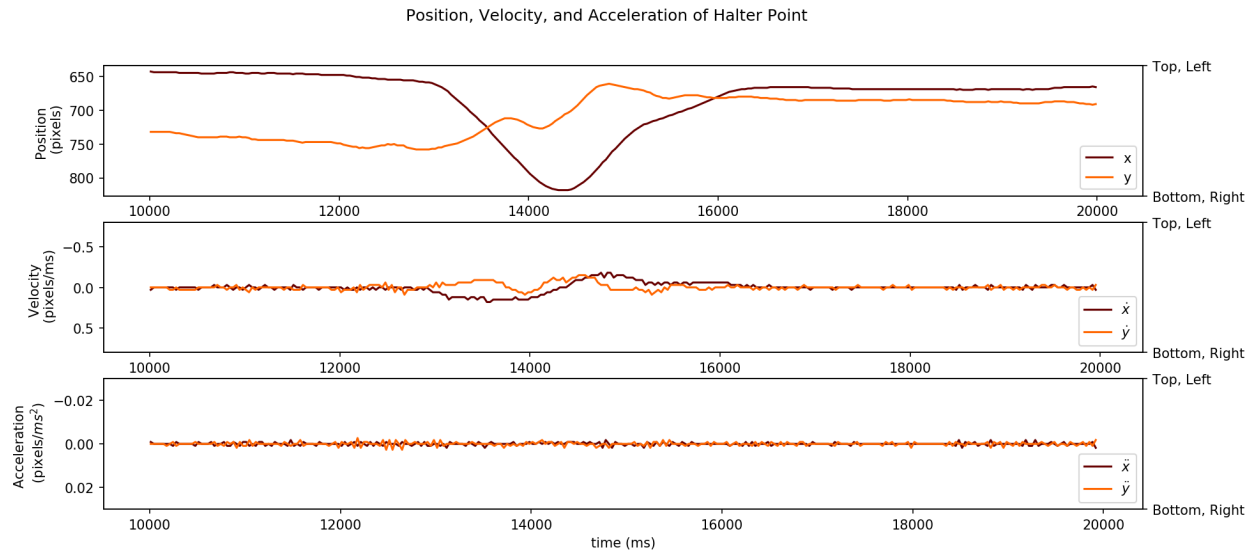


Figure 5.44: The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).

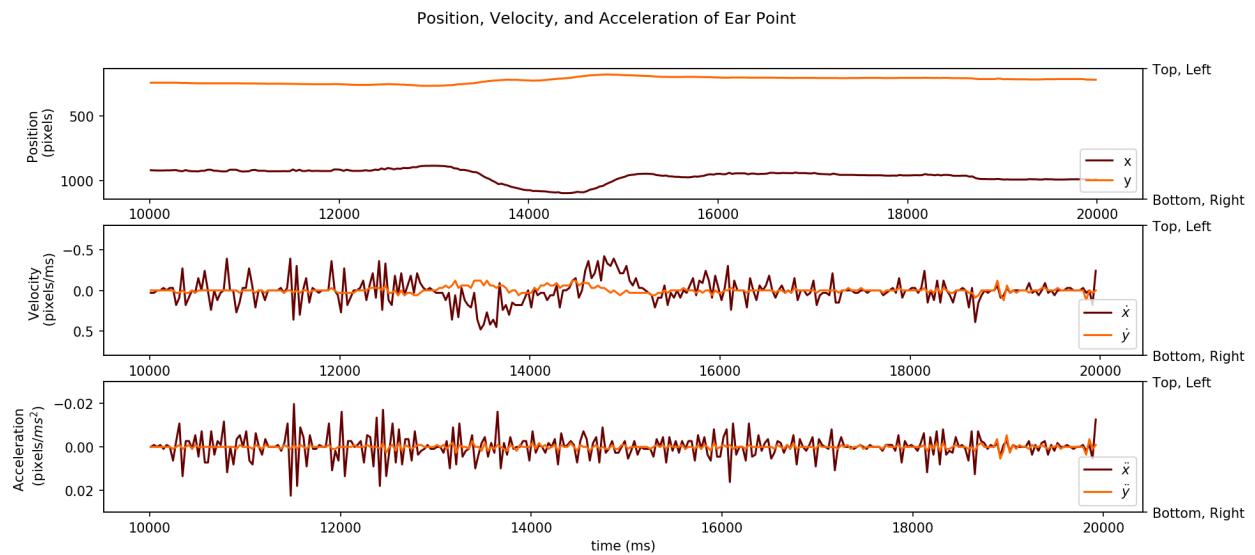


Figure 5.45: Orange is vertical motion, and maroon is horizontal motion of the ear tip.

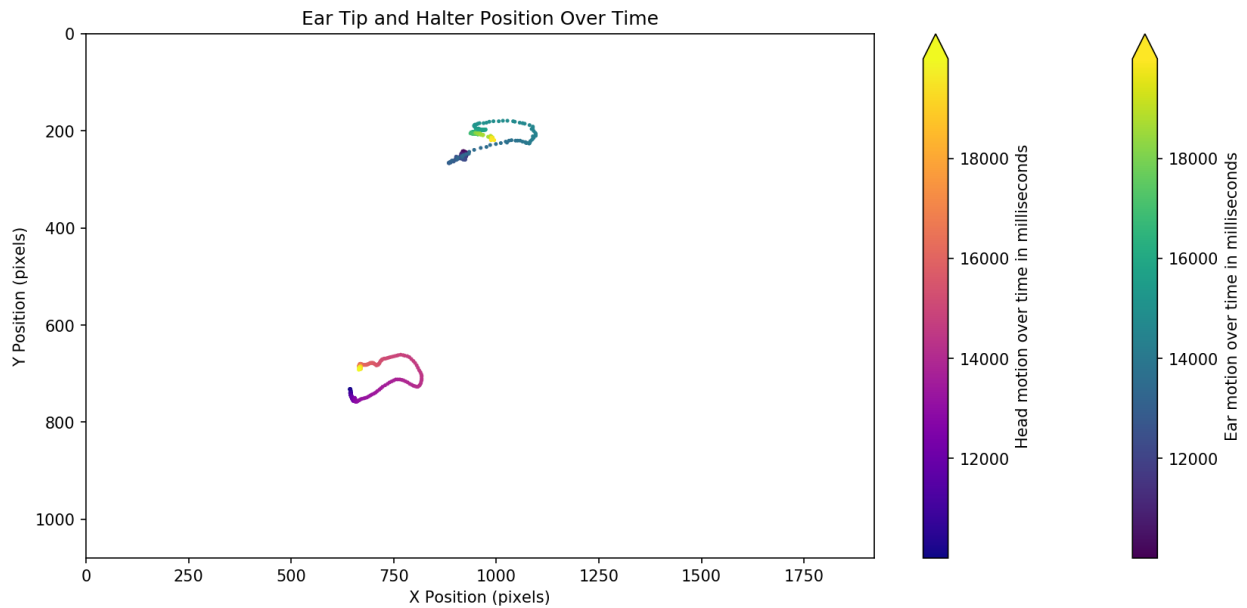


Figure 5.46: 2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicated by purple to green to yellow, and head motion is indicated by purple to red to yellow.

5.2.8 Titanic

Titanic turns to look at the camera then turns back forward. There is no ear tracking in this clip because of the purple turnout blanket.



Figure 5.47: Image of Titanic.

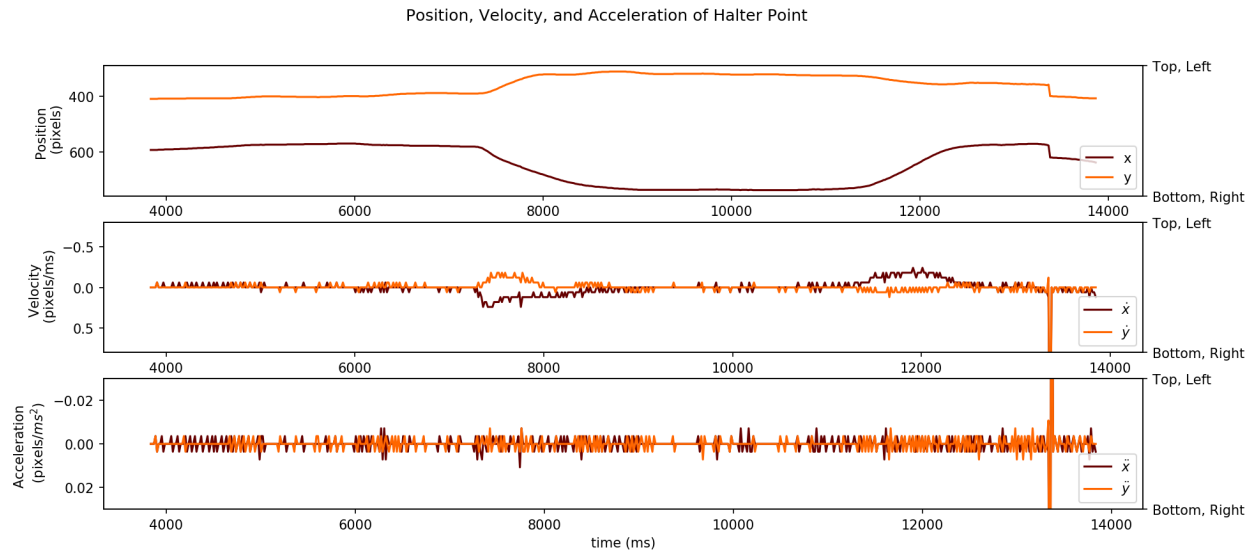


Figure 5.48: The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).

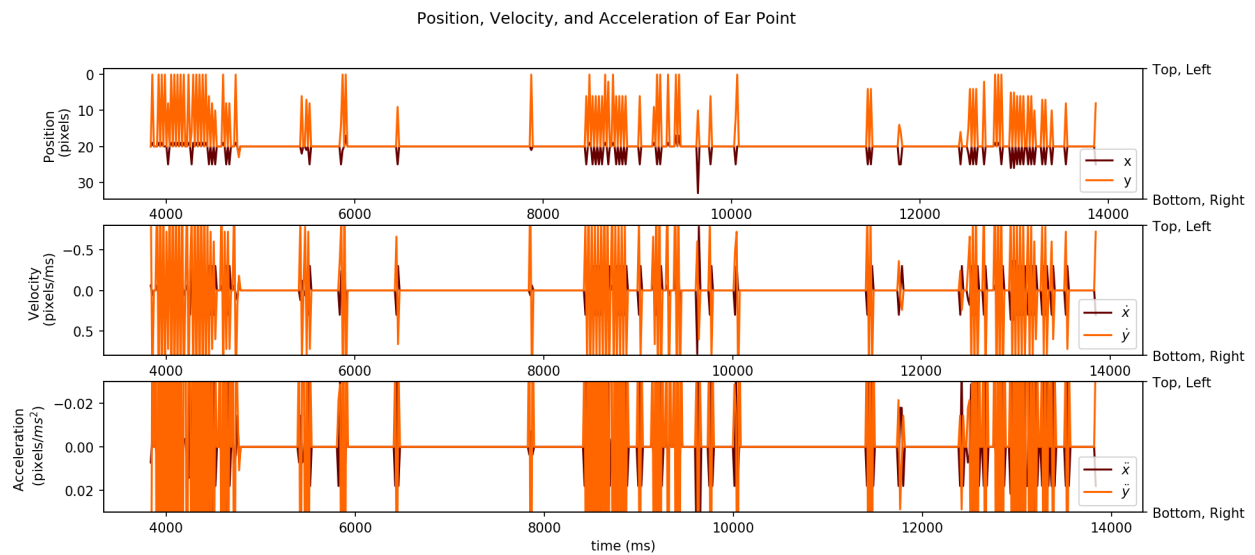


Figure 5.49: Orange is vertical motion, and maroon is horizontal motion of the ear tip.

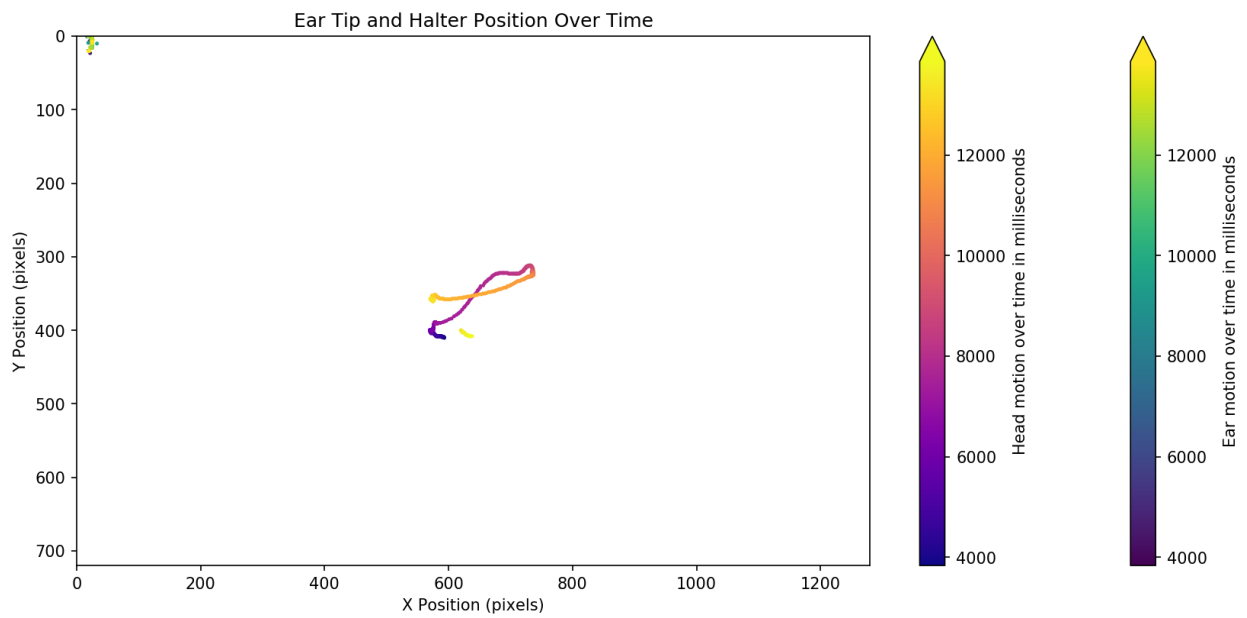


Figure 5.50: 2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicated by purple to green to yellow, and head motion is indicated by purple to red to yellow.

5.2.9 Tethys

In this clip Tehtys' ear moves forward then at the end of the clip Tehtys turns to look at the camera.



Figure 5.51: Image of Tethys.

t

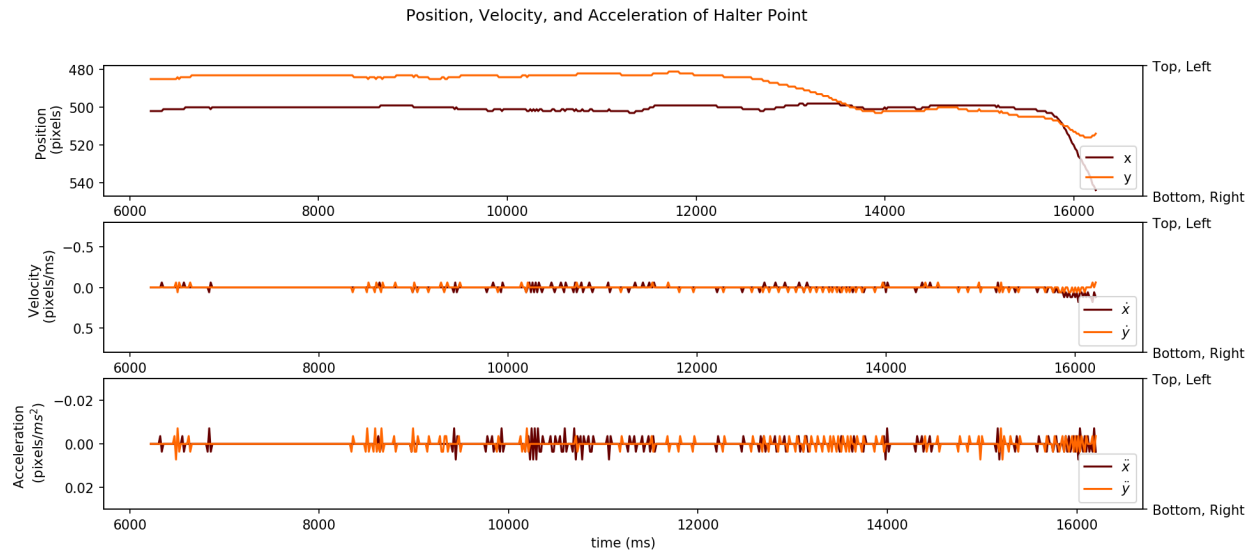


Figure 5.52: The orange line shows vertical motion, and maroon shows horizontal motion of the head(halter).

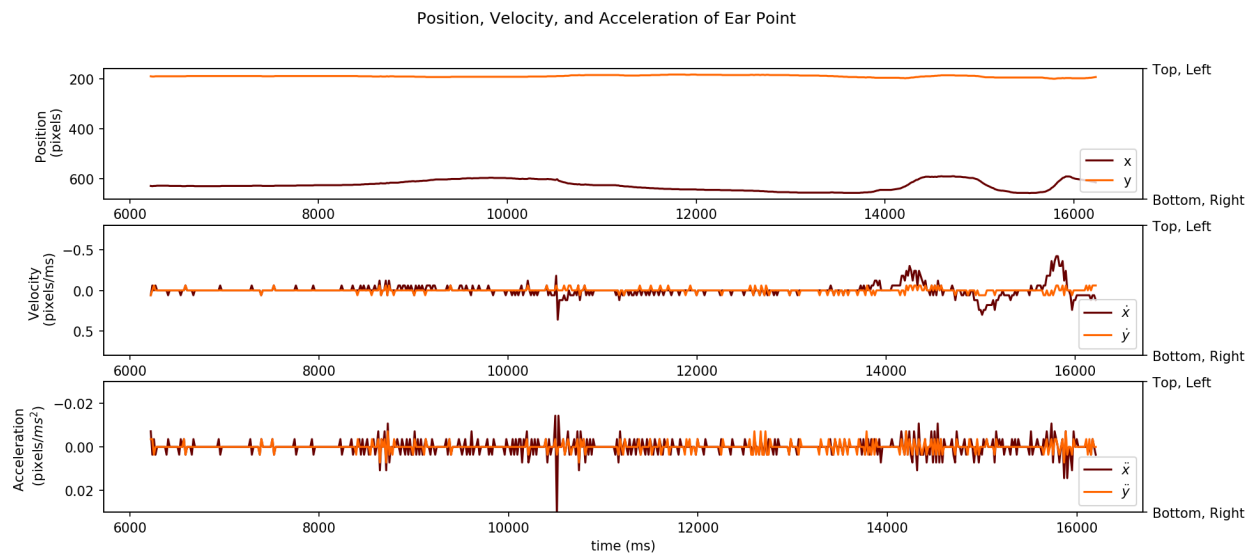


Figure 5.53: Orange is vertical motion, and maroon is horizontal motion of the ear tip.

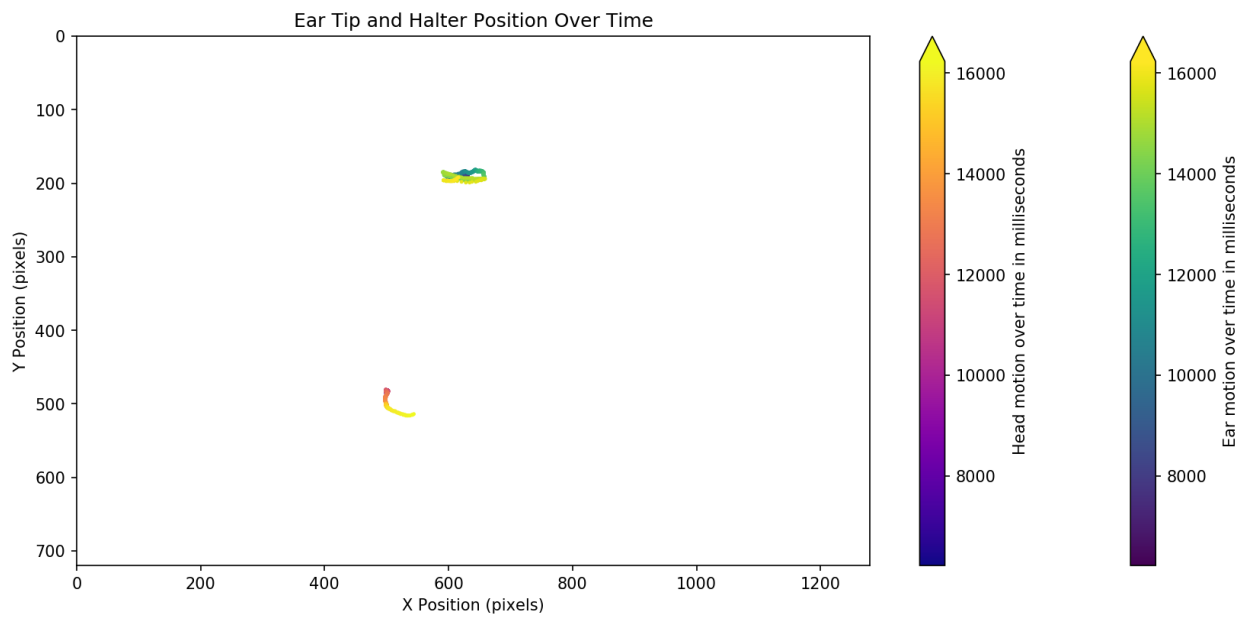


Figure 5.54: 2D motion of the head and ear together. Time is indicated by purple to yellow color change. Ear motion is indicated by purple to green to yellow, and head motion is indicated by purple to red to yellow.

5.3 Summary

The goal is to develop feature detection for horse head (halter) tracking and ear motion. This was done successfully for video scenarios of side views of horses in a field scenario at a local horse boarding facility. The horses were standing still under a shed roof and held by a handler holding a lead rope. In all cases, algorithms were run on raw video data with no human intervention. All plotted results are directly from algorithm, with only graphing parameters set manually.

The results presented here demonstrate the capability of the automatic feature tracking algorithm for tracking head(halter) motion, and ear motion.

The results demonstrate clear data trends such as quiet horse head and ear movement Figure 5.55c to a horse showing much more movement in Figure 5.55d

Data was collect in winter and some purple blankets caused issues. However these results demonstrate that with the code works well for detecting head and ear motion.

Based on these results, it is likely that position and velocity plots of feature tracking may hold promise for detecting particular patterns that may indicate pain or well-being.

Data of head and ear motion are represented numerically and shown here graphically as motion vs. time, velocity vs. time, and acceleration vs. time. This will allow researchers to look for trends and apply data analysis for determining horse condition such as pain and well-being. This work also sets the groundwork for apply conventional image processing techniques for feature detection and tracking in videos of other scenarios including many horses.

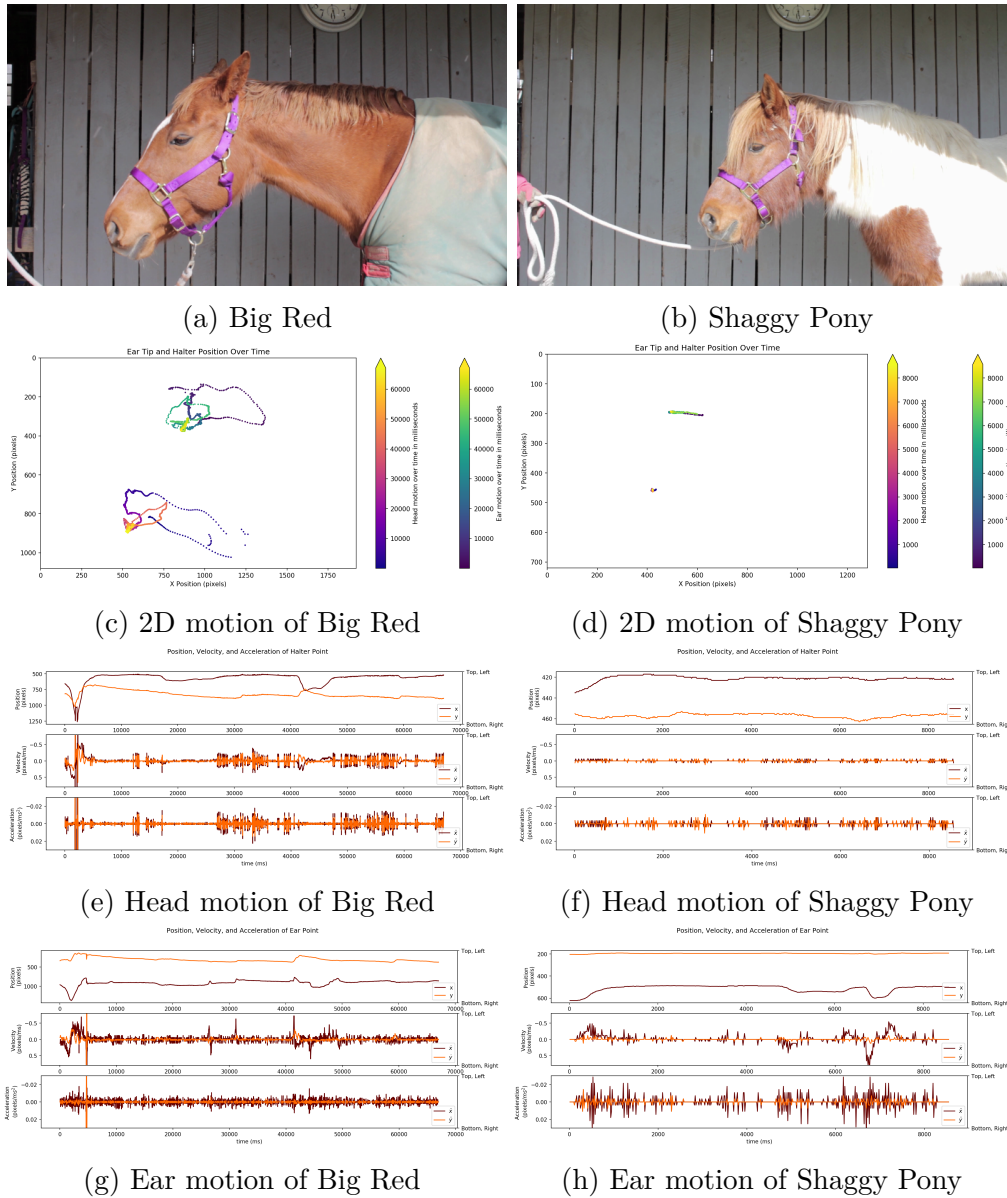


Figure 5.55: The comparison of a video with head and ear motion (left) vs. a video with just ear motion 5(right)

Chapter 6

Conclusion

The well being of horses is very important to their care takers, trainers, and owners. Detection of pain in horses is difficult because they are prey animals and have mechanisms to disguise their pain. A lot of work has been done to detect pain in horse starting with invasive methods such as heart rate monitors around the body, drawing blood for cortisol levels, or pressing on painful areas to elicit a response.

Some work has been done previously to detect horse well-being non-invasively using photographs this has included work developing and scoring with the Horse Grimace Scale by Dalla Costa et al. [4] and parallel work by Gleerup et al. [5]. Both scales require people to look at still images to detect pain on the horses face, which requires a lot of time to filter and determine the pain level in each image. Work has been done to use the previous scales on videos, but still requires people to filter and score the video clips. The work in this thesis utilizes image processing approaches to lay the foundation of automating this approach by allowing detection and tracking of halter and ear motion as well as detecting the eye on light colored horses in videos of a horse standing and looking to the left. By detecting and tracking these features, computers can be trained to detect changes in these features that are indicators of pain, stress, comfort and other indicators of well-being.

The algorithms developed in this thesis were done with the Python programming language using common open source libraries such as OpenCV. Existing subroutines available were utilized in the program. This software was developed with lower power computers and consumer level cameras in mind.

This thesis also made recommendations on things to avoid that could trip up the automatic tracking in computer vision. The recommendations are avoiding backgrounds that are similar colors, avoiding horse accessories that are a similar color to the purple halter, or avoid lighting that causes shadows on the horse. The recommendations are also areas of improvement for future work.

6.1 Contributions

The contributions of this thesis are a program that can be used animal welfare community for horses, and the first steps to tracking an animal that not a lot of work was been done for

the computer vision community.

The program for this thesis was written to work on low processing power computers, and utilize inexpensive video cameras for sensing. The program processes and displays the the output at about 2 frames per second for 1920 by 1080 pixels sized frames and 4 frames per second for 1280 by 720 pixels sized frames. At the end of the video it graphs the ear and head track and saves a comma separated value file for later processing or comparison. The program can find the open eye on light horses but does not use or display the information at this time.

Additional contributions are:

- The use of HSV as a method of background subtraction.
- A method of finding and tracking the halter.
- A method of using the halter as a landmark to find facial features on the horse.
- A way to video horses for feature tracking.
- Two methods of representing the data.
- A method for matching the halter frame by frame using ORB described in Section 3.3.

6.1.1 Recommendations

Another contribution is a list of recommendations for videoing horses. The recommendations are in no particular order.

- Make sure the background is a different color than the horse or the computer might not be able to distinguish the difference between the two.
- Avoid lighting that causes shadows, because shadows change the pixel colors, HSV can mitigate some of the changes caused by shadows but not all of them.
- Avoid ropes, blankets and other accessories that are the same color as the halter, purple for this version of code in this thesis, because the program will think they are part of the halter.
- The next two are for converting pixels into another unit like inches or millimeters.

Bring a checkerboard of know sizes to video at location, and measure distance from camera to checkerboard for camera calibration.

Measure distances of the camera to the horse, and background, because measurements from the test setup will help determine distance to locations in the video.

6.2 Future Work

This work shows a lot of potential, and has many directions it can go because of the continual research in computer vision. With more work it can be used to monitor horses post surgery, after transportation, or at competitions.

The ear and head tracking would have benefited from the use of Kalman filtering to prevent the jumping of tracked bounding boxes. Utilizing machine learning to find the eyes and determine openness and track the ears like in the Lu et al. [20] research would also be beneficial.

3D was beyond the scope of this thesis but is still important. All of the trackers only tracked in x and y and did not take depth into consideration. Tracking the 3D motion of the head would make it easier to tell if the head is turning towards or away from the camera. Having depth information will also make it easier to remove the background and determine the difference between the left and right ear. Using 3D information would help determine the difference between a head turning and a horse walking forward.

6.2.1 Areas of Improvement

This thesis tackled a new problem that exists in an unstructured environment but the problem was simplified to create a starting point, to be a foundation for future work to improve the wellbeing of horses.

A list of requirements for the program that can be expanded on. The Program:

- only works with horses facing to the left.
- only works with purple halters
- only tracks one ear.
- can not tell which ear it is tracking.
- gets confused by object similar color to the objects being tracked.

Bibliography

- [1] P. M. Taylor, P. J. Pascoe, and K. R. Mama, “Diagnosing and treating pain in the horse. where are we today?,” *The Veterinary clinics of North America. Equine practice*, vol. 18, no. 1, p. 1, 2002.
- [2] M. Stewart, T. M. Foster, and J. R. Waas, “The effects of air transport on the behaviour and heart rate of horses,” *Applied Animal Behaviour Science*, vol. 80, no. 2, pp. 143–160, 2003.
- [3] A. Tateo, B. Padalino, M. Boccaccio, A. Maggiolino, and P. Centoducati, “Transport stress in horses: Effects of two different distances,” *Journal of Veterinary Behavior: Clinical Applications and Research*, vol. 7, no. 1, pp. 33–42, 2012.
- [4] E. Dalla Costa, M. Minero, D. Lebelt, D. Stucke, E. Canali, and M. C. Leach, “Development of the horse grimace scale (hgs) as a pain assessment tool in horses undergoing routine castration,” *PloS one*, vol. 9, no. 3, p. e92281, 2014.
- [5] K. B. Glerup, B. Forkman, C. Lindegaard, and P. H. Andersen, “An equine pain face,” *Veterinary Anaesthesia and Analgesia*, vol. 42, no. 1, pp. 103–114, 2015.
- [6] K. B. Glerup and C. Lindegaard, “Recognition and quantification of pain in horses: A tutorial review,” *Equine Veterinary Education*, vol. 28, no. 1, pp. 47–57, 2016.
- [7] E. D. Costa, D. Bracci, F. Dai, D. Lebelt, and M. Minero, “Do different emotional states affect the horse grimace scale score? a pilot study,” *Journal of Equine Veterinary Science*, vol. 54, pp. 114 – 117, 2017.
- [8] S. Dyson, J. M. Berger, A. D. Ellis, and J. Mullard, “Equine research: Can the presence of musculoskeletal pain be determined from the facial expressions of ridden horses (fereq)?,” *Journal of Veterinary Behavior: Clinical Applications and Research*, vol. 19, pp. 78 – 89, 2017.
- [9] S. Dyson, J. Berger, A. D. Ellis, and J. Mullard, “Development of an ethogram for a pain scoring system in ridden horses and its application to determine the presence of musculoskeletal pain,” *Journal of Veterinary Behavior*, vol. 23, pp. 47 – 57, 2018.
- [10] M. S. Bartlett, G. Littlewort, M. Frank, C. Lainscsek, I. Fasel, and J. Movellan, “Recognizing facial expression: machine learning and application to spontaneous behavior,” in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 568–573, IEEE, 2005.

- [11] M. Bartlett, G. Littlewort, T. Wu, and J. Movellan, “Computer expression recognition toolbox,” in *2008 8th IEEE International Conference on Automatic Face Gesture Recognition*, pp. 1–2, Sep. 2008.
- [12] S. D. Roy, M. K. Bhowmik, P. Saha, and A. K. Ghosh, “An approach for automatic pain detection through facial expression,” *Procedia Computer Science*, vol. 84, pp. 99–106, 2016.
- [13] A. B. Ashraf, S. Lucey, J. F. Cohn, T. Chen, Z. Ambadar, K. M. Prkachin, and P. E. Solomon, “The painful face – pain expression recognition using active appearance models,” *Image and Vision Computing*, vol. 27, no. 12, pp. 1788–1796, 2009.
- [14] W. Commons, “File:kernel machine.png — wikimedia commons, the free media repository,” 2016. [Online; accessed 9-May-2018].
- [15] G. Littlewort, J. Whitehill, T. Wu, I. Fasel, M. Frank, J. Movellan, and M. Bartlett, “The computer expression recognition toolbox (cert),” pp. 298–305, 2011.
- [16] C. Calistra, “60 facial recognition databases,” 2015. [Online; accessed May 07, 2018].
- [17] Wikipedia contributors, “Facial expression databases — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 7-May-2018].
- [18] A. Bronkhorst, “Horse photos, horse image database.” [Online; accessed 7-May-2018].
- [19] M. Uddin and A. Akhi, “Horse detection using haar like features,” *International Journal of Computer Theory and Engineering*, vol. 8, no. 5, p. 415, 2016.
- [20] Y. Lu, M. Mahmoud, and P. Robinson, “Estimating sheep pain level using facial action unit detection,” in *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*, pp. 394–399, May 2017.
- [21] H. Yang, R. Zhang, and P. Robinson, “Human and sheep facial landmarks localisation by triplet interpolated features,” *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, p. 1, 2016.
- [22] S. G. Sotocinal, R. E. Sorge, A. Zaloum, A. H. Tuttle, L. J. Martin, J. S. Wieskopf, J. C. Mapplebeck, P. Wei, S. Zhan, S. Zhang, J. J. McDougall, O. D. King, and J. S. Mogil, “The rat grimace scale: A partially automated method for quantifying pain in the laboratory rat via facial expressions,” *Molecular Pain*, vol. 7, p. 55, Jul 2011.
- [23] P. S. Foundation, “The python tutorial: Whetting your appetite.” <https://docs.python.org/3/tutorial/appetite.html>. [Online; accessed 16-October-2018].
- [24] “Opencv.” <https://opencv.org/>. [Online; accessed Apr. 21, 2018].

- [25] OpenCV, “Matbasicimageforcomputer.jpg.” https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html#py-display-image. [Online; accessed Nov. 14, 2017].
- [26] call_me_yang, “Using matplotlib with python.” <https://blog.csdn.net/yang6464158/article/details/22215695e>. [Online; accessed Apr. 26, 2018], Title translated using Google Translate.
- [27] OpenCV, “File:border.jpg.” [Online; accessed 5-December-2018].
- [28] SharkD, “Hsv color solid cylinder alpha lowgamma.” https://en.wikipedia.org/wiki/File:HSV_color_solid_cylinder_alpha_lowgamma.png#filelinks. [Online; accessed Apr. 21, 2018].
- [29] Jacobolus, “Unintuitive-rgb.png.” https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:Unintuitive-rgb.png. [Online; accessed Apr. 26, 2018].
- [30] A. Koschan and M. Abidi, “Detection and classification of edges in color images,” *IEEE Signal Processing Magazine*, vol. 22, pp. 64–73, Jan 2005.
- [31] “File:hysteresis.jpg.” [Online; accessed 23-Dec-2018].
- [32] “Contours : Getting started.” https://docs.opencv.org/3.2.0/d4/d73/tutorial_py_contours_begin.html. [Online; accessed Apr. 23, 2018].
- [33] “Orb (oriented fast and rotated brief).” https://docs.opencv.org/3.2.0/d1/d89/tutorial_py_orb.html. [Online; accessed 23-Dec-2018].
- [34] Goffrie, “Hsv-rgb-comparison.svg.” <https://en.wikipedia.org/wiki/File:HSV-RGB-comparison.svg>. [Online; accessed Apr. 21, 2018].
- [35] W. Commons, “File:halterparts.png — wikimedia commons, the free media repository,” 2017. [Online; accessed 7-October-2018].
- [36] “Ubuntu 16.04: How to install opencv.” [Online; accessed June,16,2017].

Appendices

Appendix A

Release Form



Virginia Polytechnic Institute and State University
Veterinary Teaching Hospital
Address: 245 Duck Pond Dr., Blacksburg, Virginia 24061-0443
Phone: 540-231-4621 | Fax: 540-231-9354

Clinical Research Project Client Consent Form

Study Title: Development of Computer Vision Algorithms for Detection of Horse Facial Features

Principal Investigator: Dr. Mary Kasarda, VT Department of Mechanical Engineering
540-231-8552, maryk@vt.edu

One of the missions of the Virginia-Maryland College of Veterinary Medicine is to create, disseminate and apply medical knowledge through discovery, learning, and engagement. You are invited to participate in this mission by enrolling your animal in a clinical research study. Your participation is entirely voluntary, and you may withdraw your animal from the study at any time by notifying the Principal Investigator. There is no penalty if you choose not to participate.

Study Purpose:

Researchers from the Virginia Tech (VT) Department of Mechanical Engineering and the Virginia-Maryland College of Veterinary Medicine are working together to develop computer vision/image processing algorithms to automatically detect facial features of horses in videos. This is the foundation work for developing techniques to automatically detect indicators of stress and pain in horses while undergoing continuous video monitoring. This technology will allow for more continuous monitoring of critically horses and mares in foal, and also provide more accurate assessment of the horse's response to training methods, types of competition, and other activities where they perform and interact with humans. As a first step, the technical goal of this current study is to develop image processing algorithms that can automatically detect ears, eyes, nostrils, and head angle of horses, and automatically track their movement, in videos. In order to accomplish this, researchers will need to obtain standard video images of multiple horses standing at rest in cross-ties (or being held quietly by a lead rope). Video images from a large number of horses are needed to develop computer vision algorithms, or computer codes, that will reliably work for a wide variety of horse conformations and colors. Videos and still images from these tests will be included in research publications and presentations as well as included in a public data base for use by other researchers to further advance these techniques. At no time will the actual horse name, owner name, or farm name be used or made public.

Study Design/Procedures:

Brief, lay language overview of design and procedures. Describe assignment to study groups, duration of study and frequency of visits. Indicate if study participation involves withholding of standard treatment. Describe in lay terms the conditions that would prompt discontinuation of the clinical trial protocol.

Design and Procedure of Tests. A standard camera on a tripod will be used to obtain video data for the study. Once the camera is set up, a 5-10 second video of the filming location will be taken to record an image of the background. Next, horses will be brought into view of the camera so that a video of the side view of the horse from shoulders up to the head of individual horses will be taken while the horse standing quietly in cross-ties or held by a lead rope (if this is preferred by the horse owner). Each horse will need to wear a brightly-colored standard nylon halter (provided by the research team). Once the horse is positioned in cross-ties (or held by lead rope) in front of the camera, approximately 1-3 minutes of video will be recorded. In some cases, small circular low-adhesive stickers (approximately 1 inch in diameter) may be placed on a horse's neck, head, and/or ears for additional test video footage. These stickers are non-toxic, easily removed by hand, and leave no residue on the horse. The same video recording procedure will take place, with approximately 1-3 minutes of video recorded for the side view of the horse from shoulders up, wearing a brightly-colored halter and standing in cross-ties (or held by lead rope). Upon completion of filming, stickers will be removed (if used), and halters will be collected by researchers and study is complete.

Assignment to Study Groups. All horses are included in the same study group.

Duration of Tests and Frequency of Visits. At this point in the study, we anticipate only one visit for filming of a horse will be needed. The duration of the actual test, including placement of halter on horse, positioning of horse, video recording,

and removal of halter from horse is expected to last under 10 minutes for each horse. If a second video is recorded with stickers placed on the horse's neck, head, and/or ears, this is expected to add a maximum of 10 minutes to the test, with a resulting total duration of under 20 minutes from start to finish for the recording of both videos.

Discontinuation of the test protocol: If a horse becomes clearly agitated with any aspect of the test procedure, and/or at the request of the owner, the test will cease.

Risks and Benefits:

Lay description of corresponding IACUC section (if available), including risks of withholding standard treatment if applicable.

Risks may include agitation of some horses by the presence of camera and tripod or by attempts to apply stickers to head, neck, and ears of the horse. Since no veterinary interventions or treatment are applied, risk associated with administering a treatment or withholding a treatment is not applicable. Benefits include knowledge that participation in the study supports efforts to develop novel automatic video techniques that can ultimately lead to improving the welfare of horses and other animals.

Study Costs and Compensation:

If applicable. Compensation can be combined with the Benefits section above. List any costs not covered by the study. There is no compensation for participating in this study.

Confidentiality:

The data collected in the course of this study is confidential. In any publication or presentation of the study data, we will not include information that would make it possible to identify a research participant. Research records will be kept in a secure location; only researchers will have access to the records. *Since there is no sponsor of this project, only VT researchers will have access to full information associated with the data.*

Statement of Consent:

In giving my consent by signing this form, I acknowledge that I have been informed of the purpose and nature of this study and its associated procedures, as well as any possible side effects.

I have read and understood the above information. I have been given the opportunity to ask questions and receive answers, and I consent to participate in the study. I further certify that I am the owner (or duly authorized agent of the owner) of _____ .
(Animal's name)

Owner or Agent Signature: _____ Date: _____

Attending Clinician/Researcher Signature: _____ Date: _____

Please don't hesitate to contact us if you have any questions or concerns about this study.

The research and procedures have been reviewed and approved by *OPTIONAL: the Virginia Tech Institutional Animal Care and Use Committee* and the Virginia-Maryland College of Veterinary Medicine Veterinary Teaching Hospital Board.

If you have any questions or concerns regarding the study and would like to talk to someone other than the researchers, please contact:

Hospital Director,
Veterinary Teaching Hospital
Virginia-Maryland College of Veterinary Medicine
Address: 245 Duck Pond Dr., Blacksburg, Virginia 24061-0443
Phone: 540-231-4621

You will be given a copy of this form to keep for your records.

Appendix B

Video Data

This chapter shows all the the raw data organized by videos and clips. Each section is the video title and details the horse in the video, the frame size and frame rate. If the video was not used a short description is given explaining why it was not used.

B.1 mvi_4891.mov

horse: White Star

did not use, too much camera movement, and camera was later moved to different position.

B.2 mvi_4892.mov

horse: Big Red

did not use, too much camera movement, and camera was later moved to different position.

B.3 mvi_4893.mov

Horse: Big Red

Frame size: 1920 x 1080 pixels

Frame rate: 29.97 fps

B.3.1 edit 00

- head turns to look at camera
- ear moves at frame 798 forward then back head is still



Figure B.1: Large image of Big Red.

- ear and head move at frame 1258

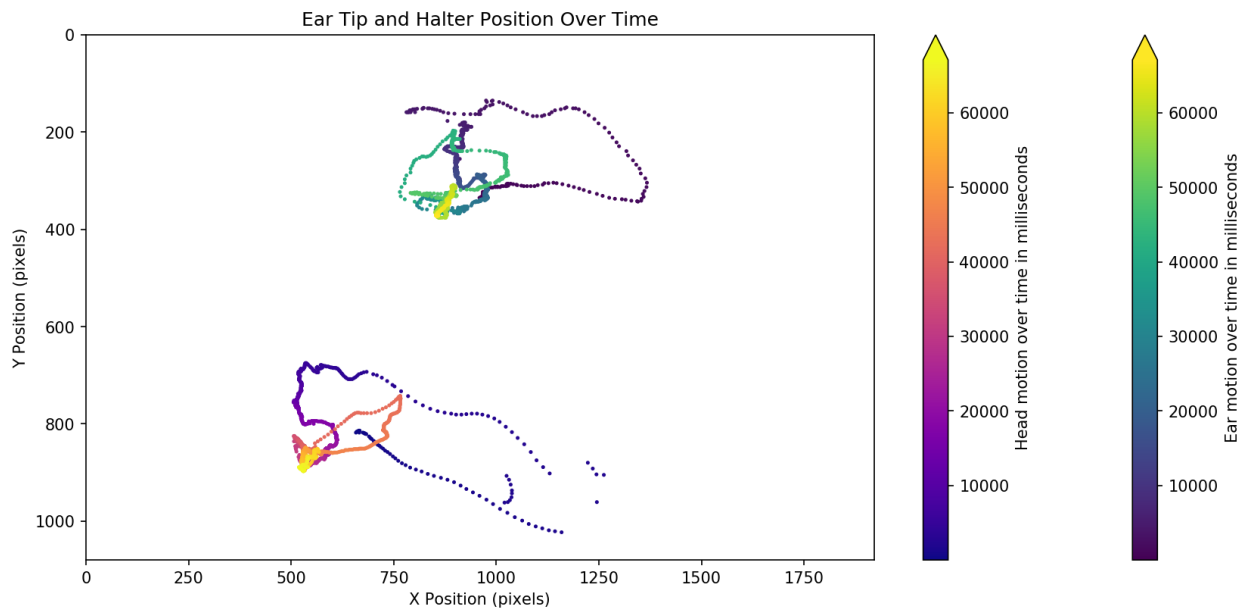


Figure B.2: 2D Head and ear motion with color changing with respect to time.

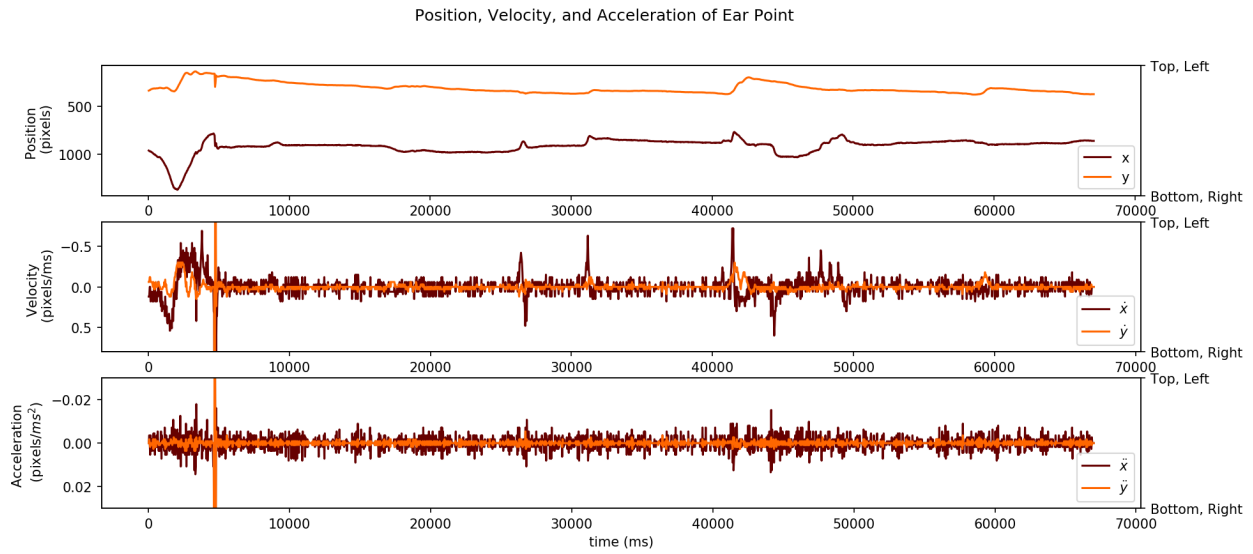


Figure B.3: Graph of ear position, velocity and acceleration. Orange is vertical motion, and maroon is horizontal motion.

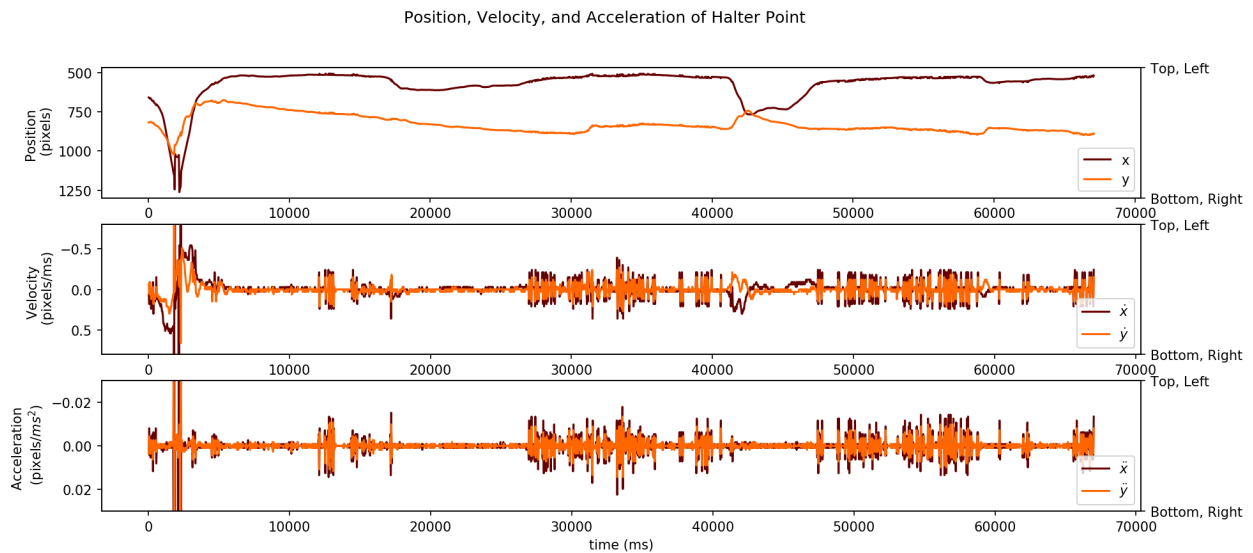


Figure B.4: Graph of head position, velocity and acceleration through halter point. Orange is vertical motion, and maroon is horizontal motion.

B.3.2 edit 01

- mostly still
- ear forward at frame 800
- alert at fr 1195 toward camera

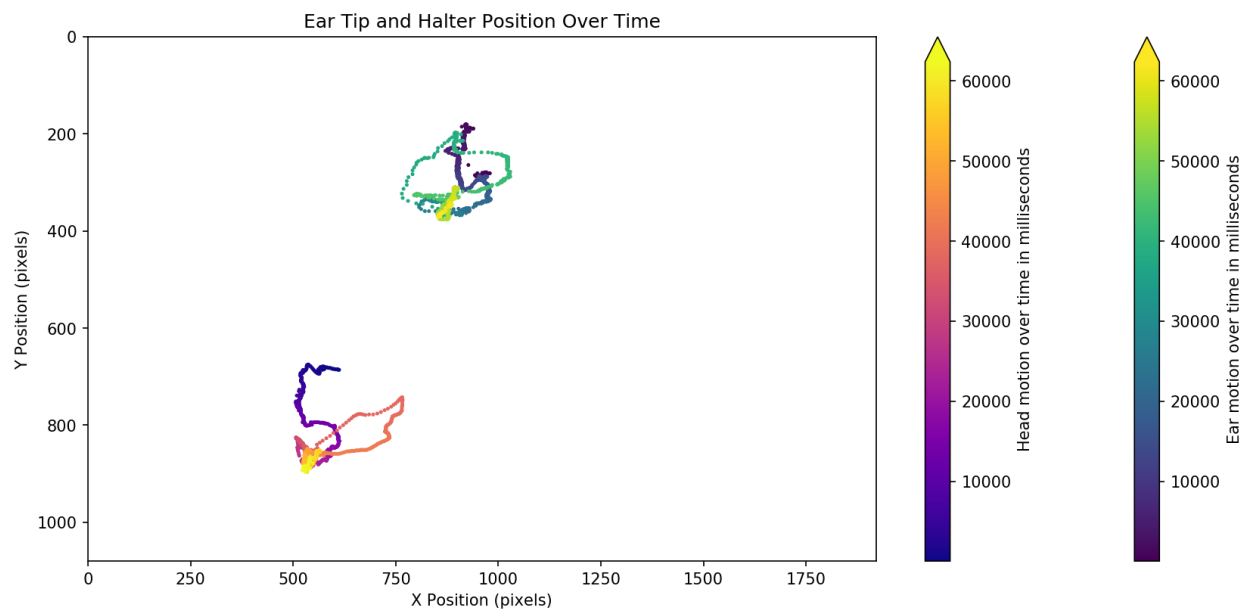


Figure B.5: 2D Head and ear motion with color changing with respect to time.

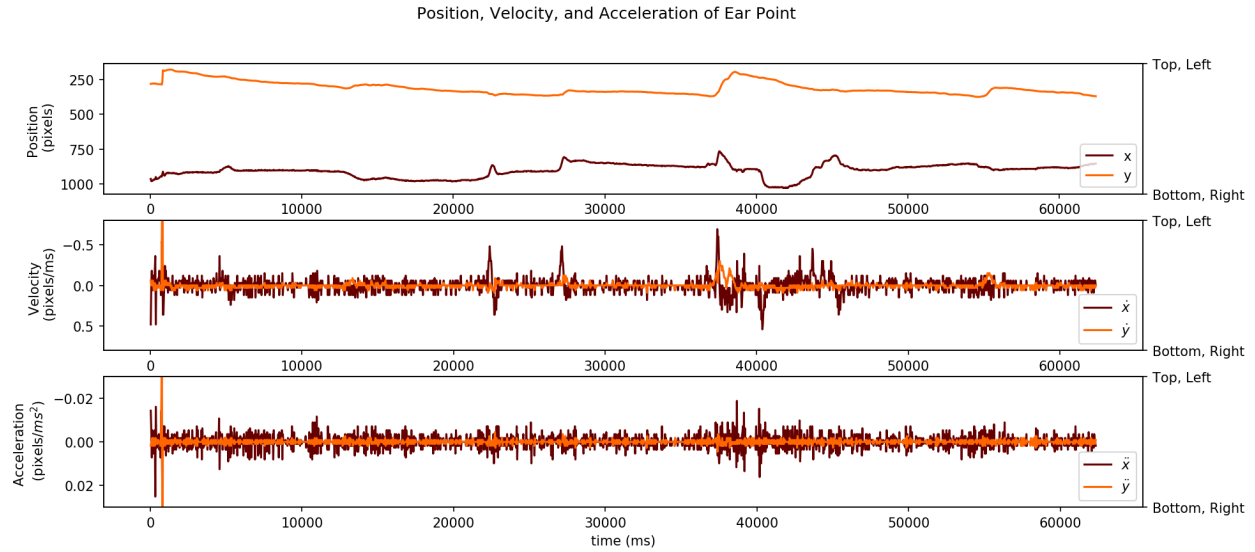


Figure B.6: Ear position, velocity and acceleration.

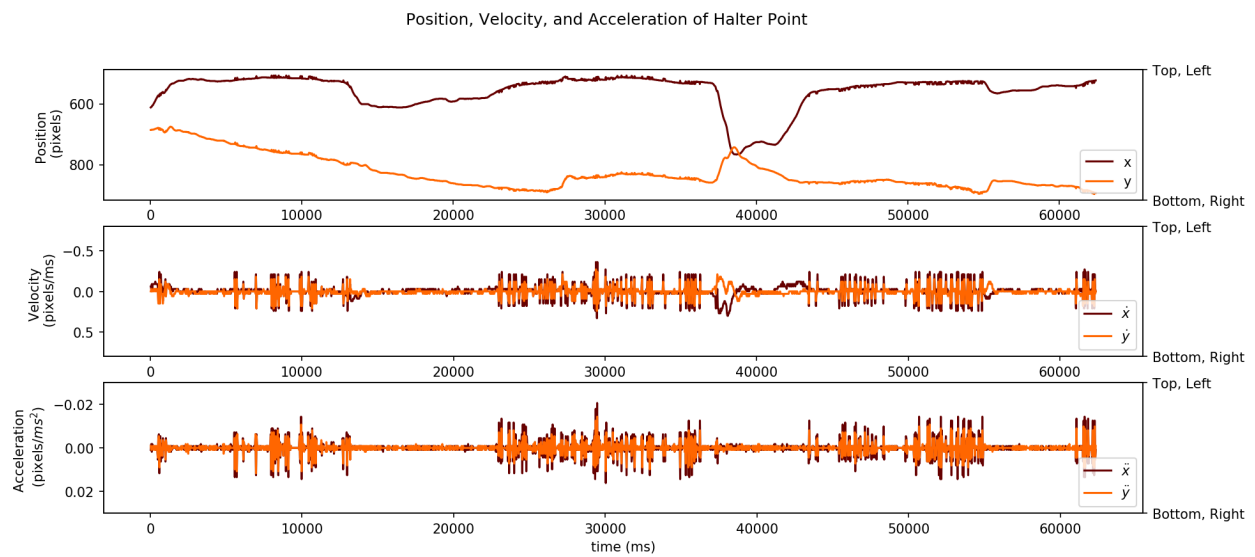


Figure B.7: Ear movement measurements.

B.4 mvi_4894.mov

Horse: White Star

Frame size: 1920 1080 pixels

Frame rate: 29.97 fps

B.4.1 edit 01

- looks at camera
- ear moves forward at fr 106
- back then forward



Figure B.8: White Star

B.4.2 edit 02

- looks at camera

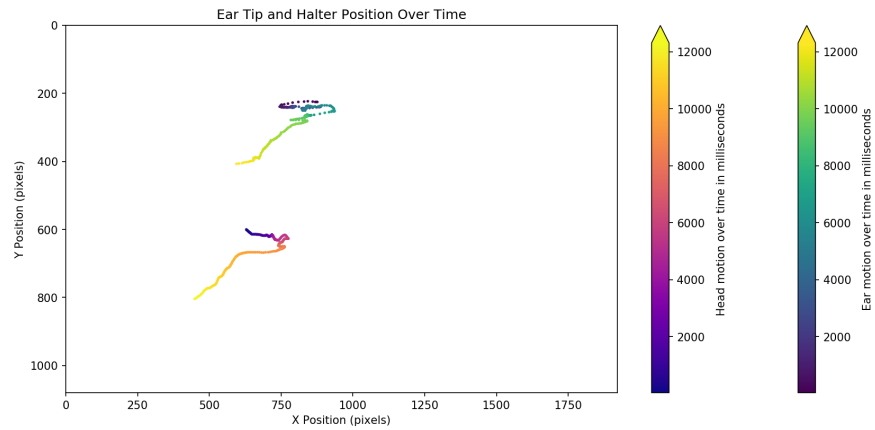


Figure B.9: 2D Head and ear motion with color changing with respect to time.

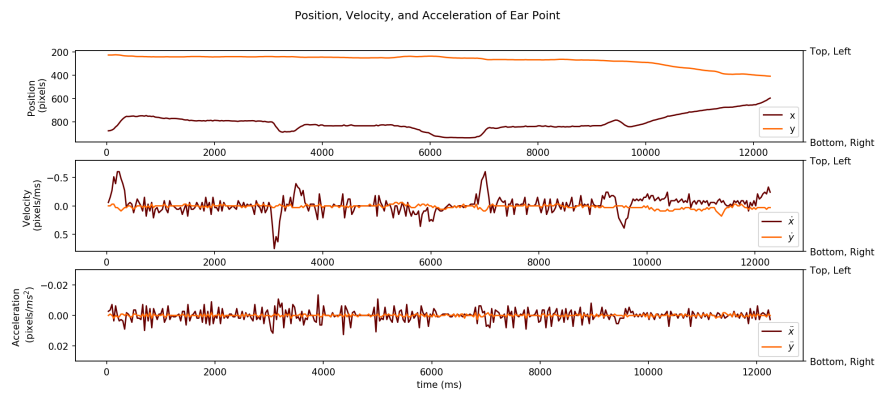


Figure B.10: Ear position, velocity and acceleration.

- bad ear track

B.4.3 edit 03

- really bad ear track
- starts ear fr 225
- loses 405

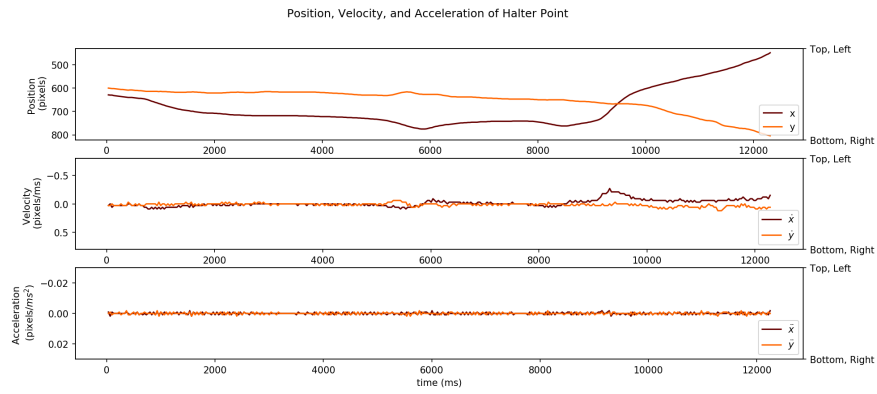


Figure B.11: Ear movement measurements.

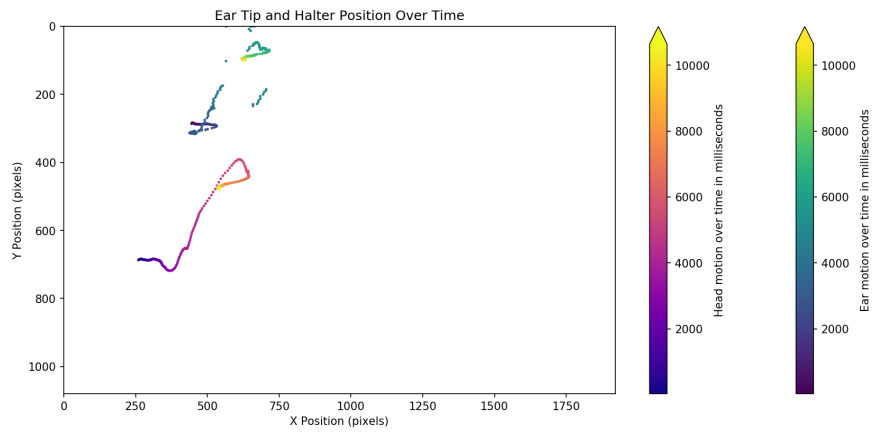


Figure B.12: 2D Head and ear motion with color changing with respect to time.

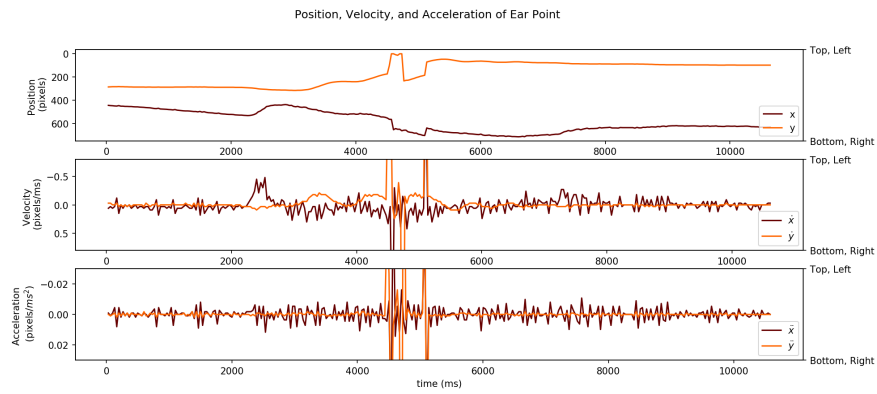


Figure B.13: Ear position, velocity and acceleration.

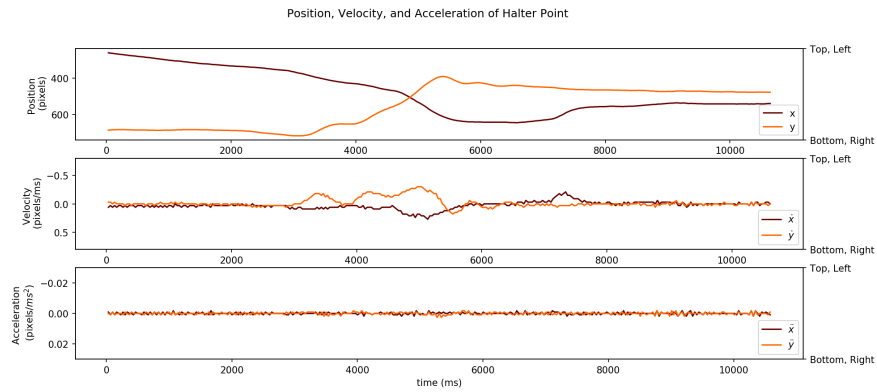


Figure B.14: Ear movement measurements.

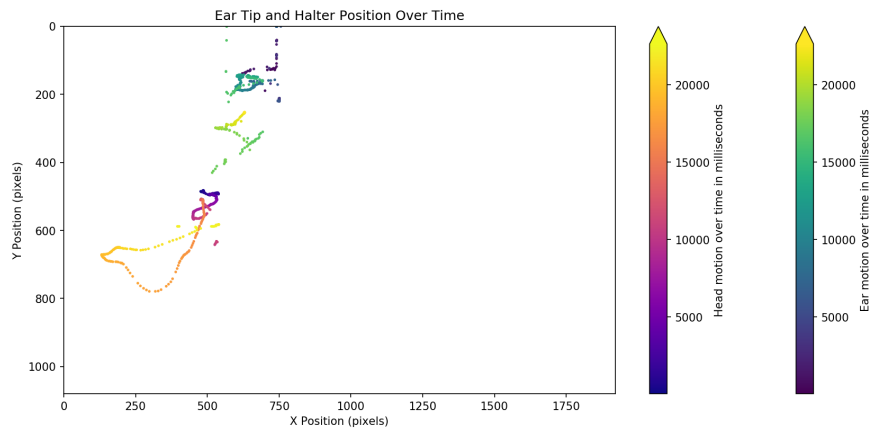


Figure B.15: 2D Head and ear motion with color changing with respect to time.

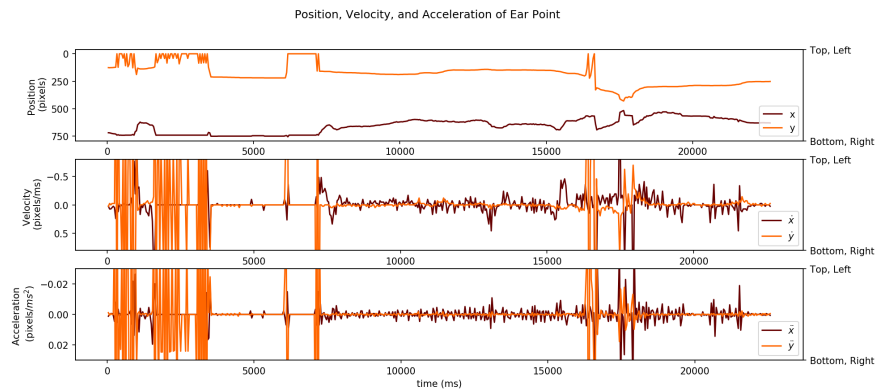


Figure B.16: Ear position, velocity and acceleration.

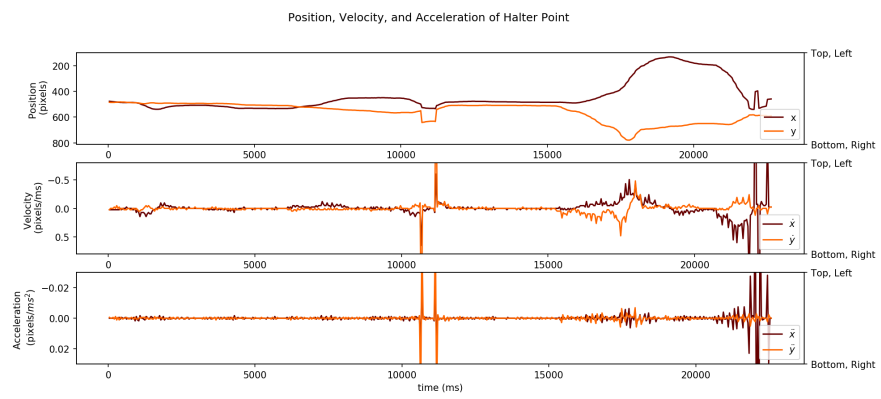


Figure B.17: Ear movement measurements.

B.5 mvi_4895.mov

Horse: White Star

Frame size: 1280 x 720 pixels

Frame rate: 59.94 fps

B.5.1 edit 00

- head down
- bad ear track
- head starts moving up fr 407 stops fr 523
- head starts down fr 1293 stops fr 1400 and forward

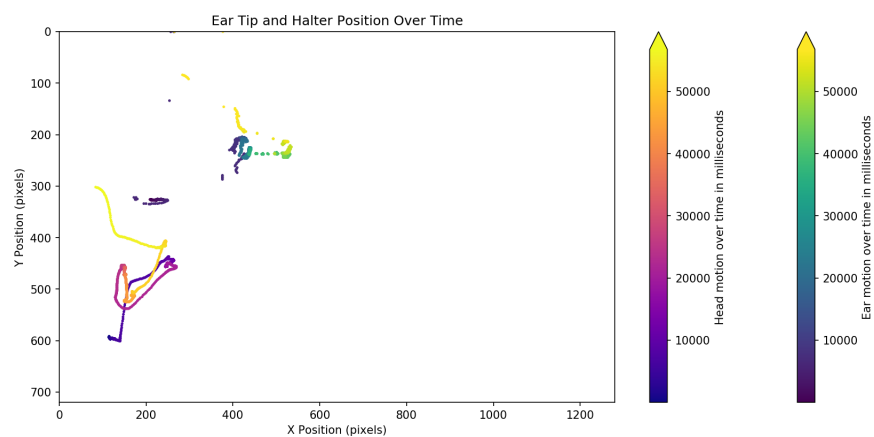


Figure B.18: 2D Head and ear motion with color changing with respect to time.

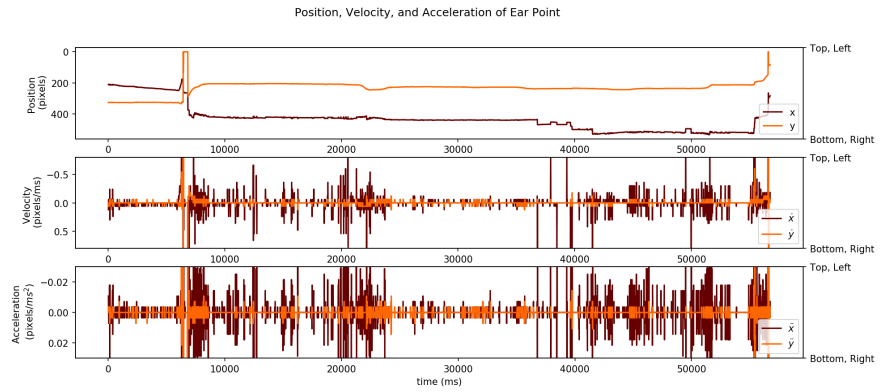


Figure B.19: Ear position, velocity and acceleration.

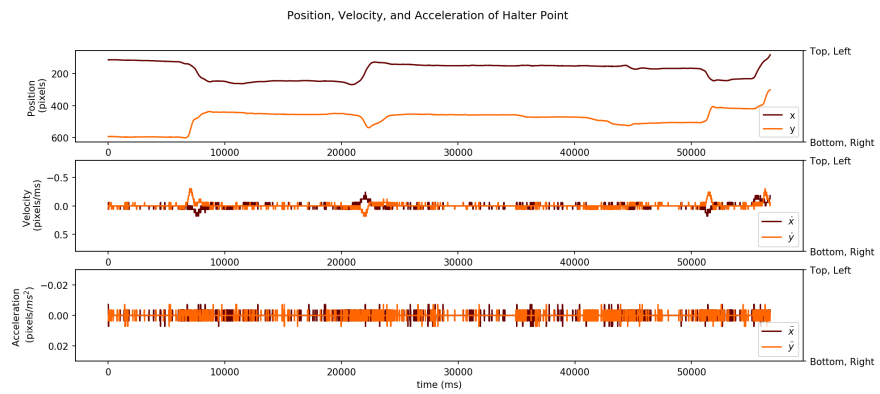


Figure B.20: Ear movement measurements.

B.6 mvi_4896.mov

Background video no horse.

B.7 mvi_4897.mov

Background video no horse.

B.8 mvi_4898.mov

Horse: Big Benny

Frame size: 1280 x 720 pixels

Frame rate: 59.94 fps

- looks away from camera
- purple blanket
- looks toward camera



Figure B.21: Big Benny

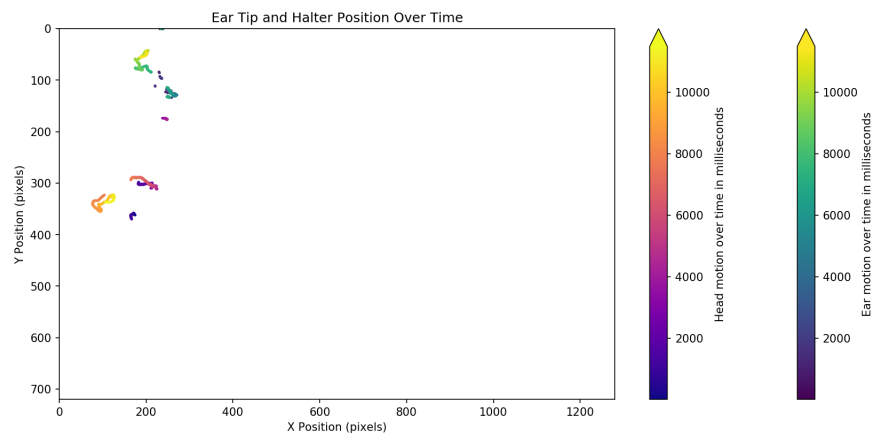


Figure B.22: 2D Head and ear motion with color changing with respect to time.

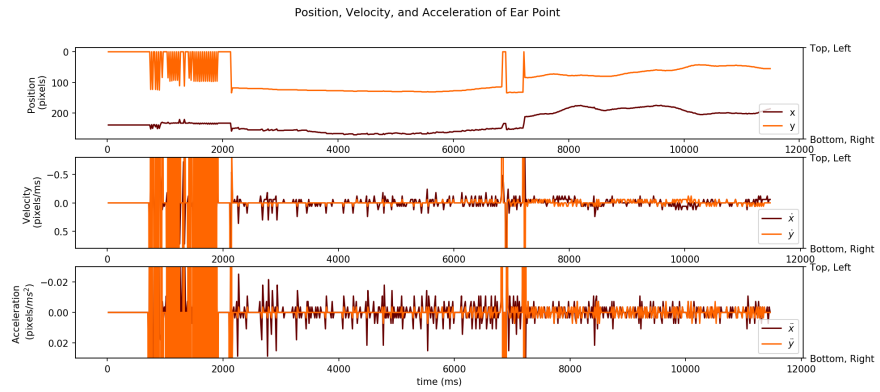


Figure B.23: Ear position, velocity and acceleration.

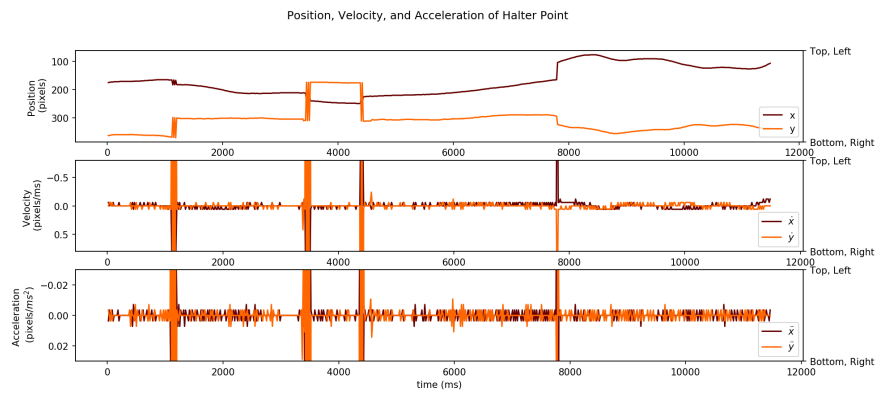


Figure B.24: Ear movement measurements.

B.9 mvi_4899.mov

Horse: Big Benny

Frame size: 1920 x 1080 pixels

Frame rate: 29.97 fps

- purple blanket
- shaky head tracking
- no ear tracking

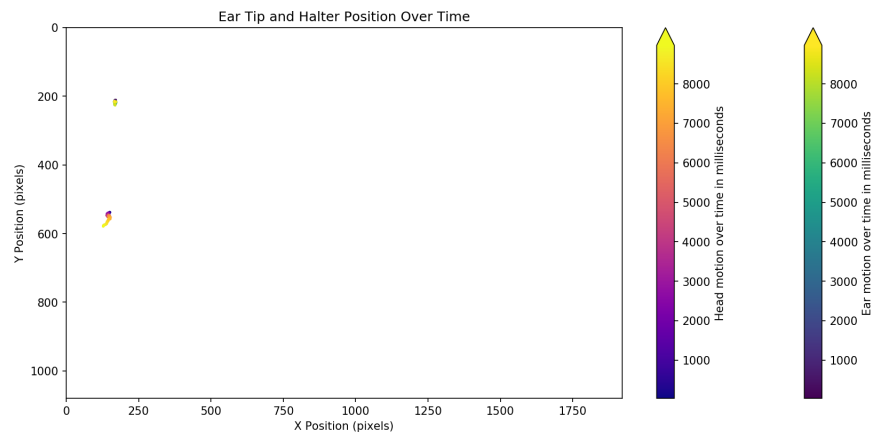


Figure B.25: 2D Head and ear motion with color changing with respect to time.

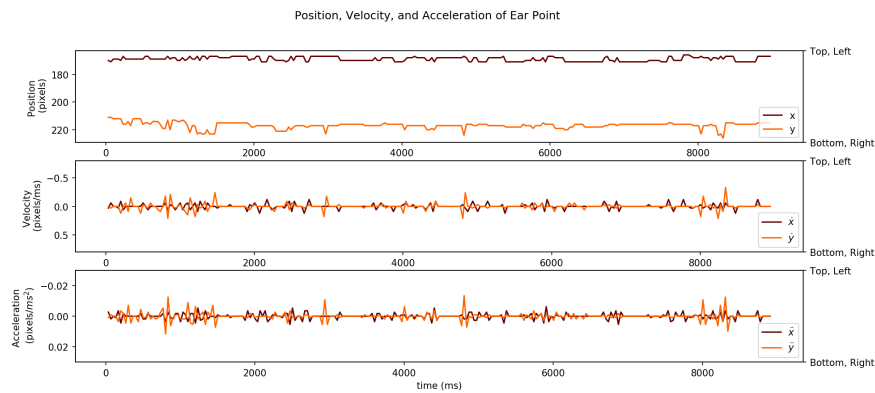


Figure B.26: Ear position, velocity and acceleration.

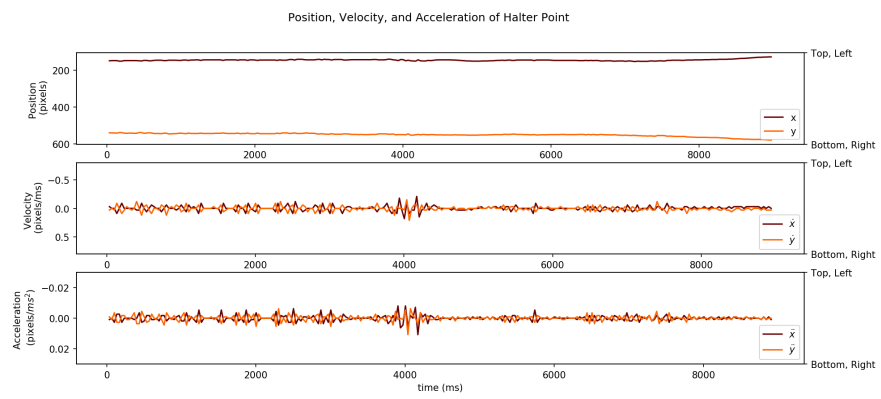


Figure B.27: Ear movement measurements.

B.10 mvi_4900.mov

Horse: Chief

Frame size: 1920 x 1080 pixels

Frame rate: 29.97 fps

- still horse
- looks away from camera at fr 317



Figure B.28: Chief

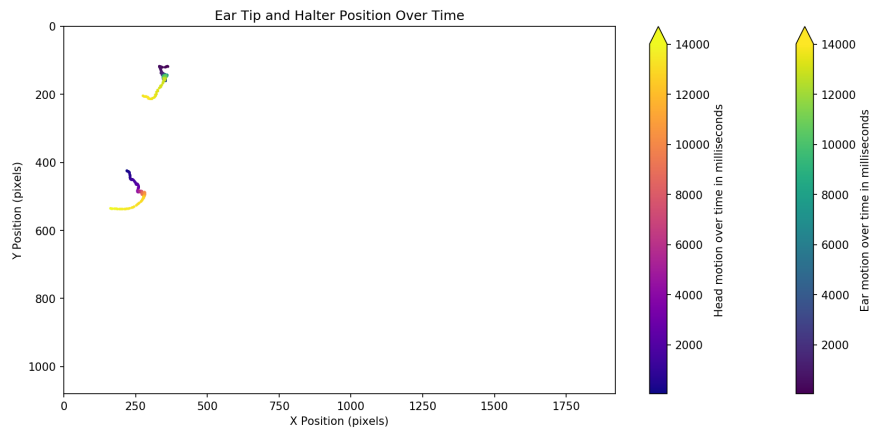


Figure B.29: 2D Head and ear motion with color changing with respect to time.

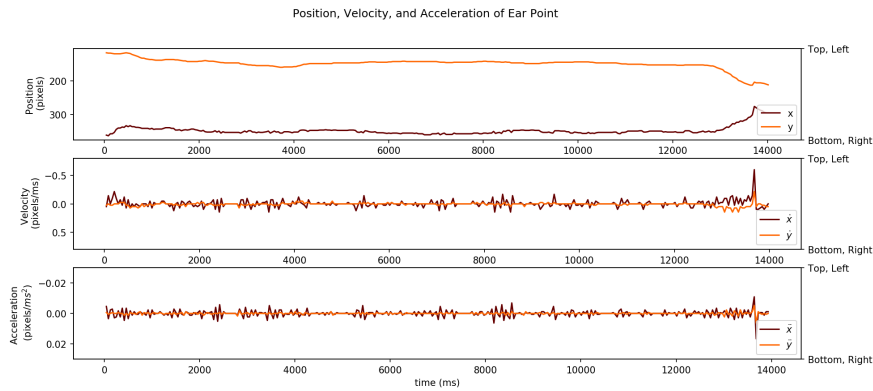


Figure B.30: Ear position, velocity and acceleration.

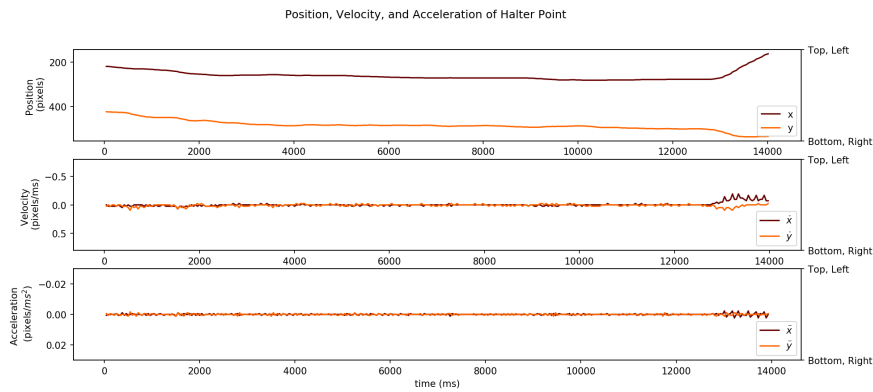


Figure B.31: Ear movement measurements.

B.11 mvi_4901.mov

Horse: Chief

Frame size: 1280 x 720 pixels

Frame rate: 59.94 fps

- doesn't move much
- ear moves a little bit

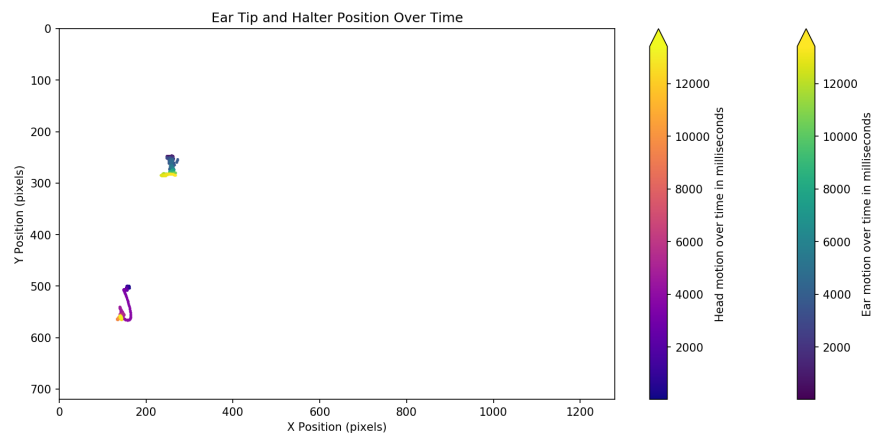


Figure B.32: 2D Head and ear motion with color changing with respect to time.

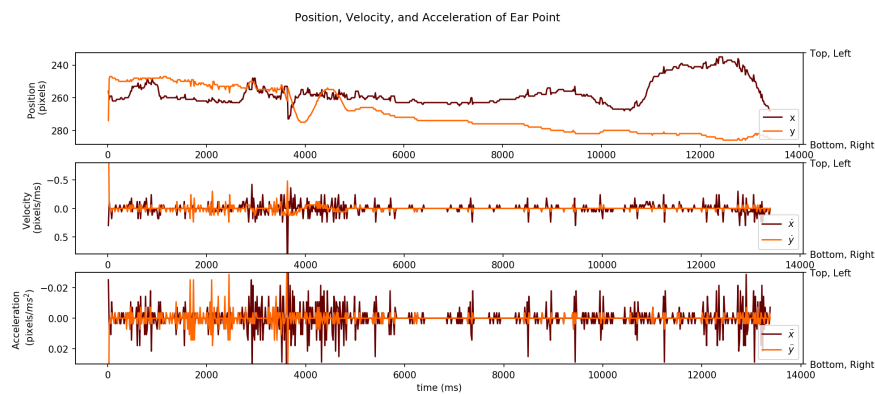


Figure B.33: Ear position, velocity and acceleration.

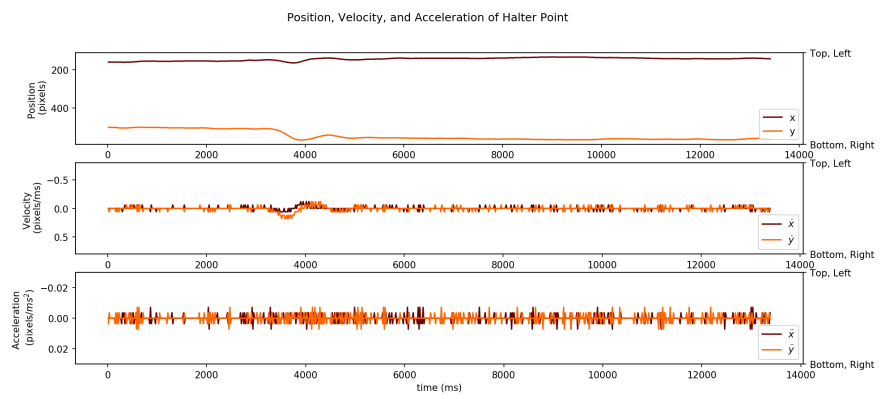


Figure B.34: Ear movement measurements.

B.12 mvi_4902.mov

Horse: Mosfet

Frame size: 1280 x 720 pixels

Frame rate: 59.94 fps

- ear moves



Figure B.35: Mosfet

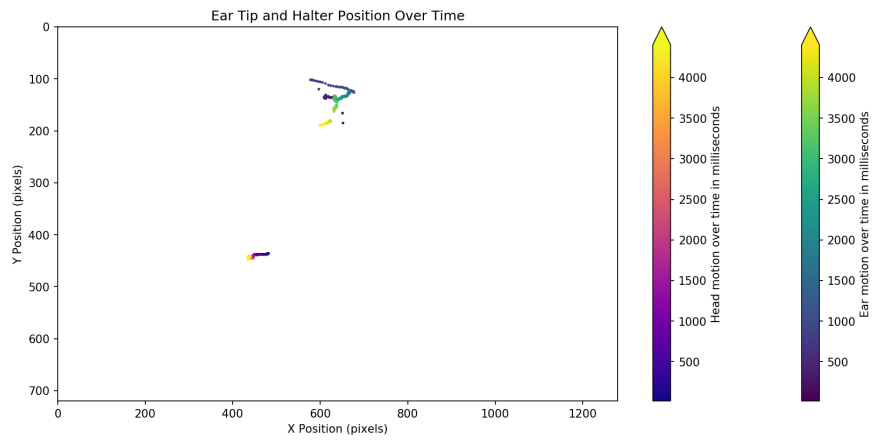


Figure B.36: 2D Head and ear motion with color changing with respect to time.

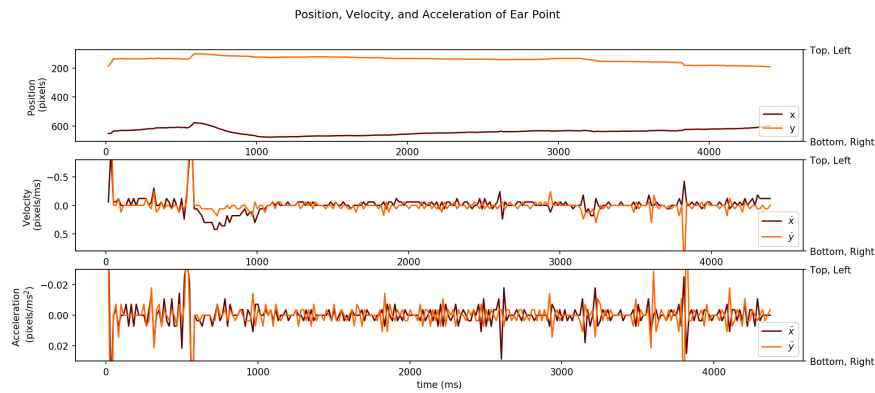


Figure B.37: Ear position, velocity and acceleration.

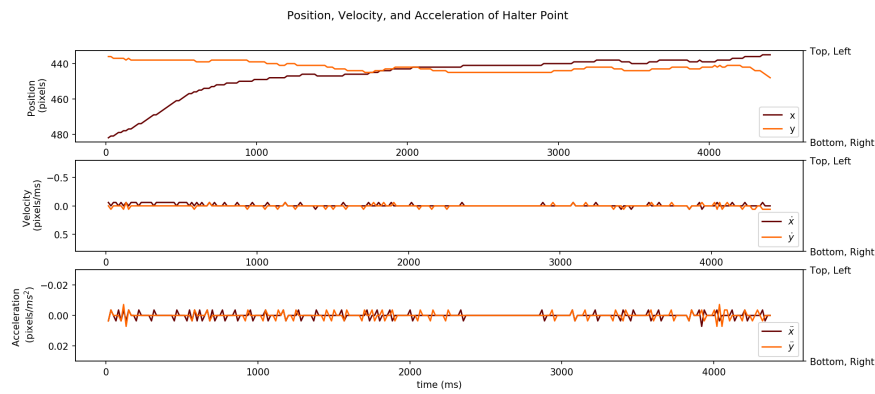


Figure B.38: Ear movement measurements.

B.13 mvi_4903.mov

Horse: Mosfet

Frame size: 192 x 1080 pixels

Frame rate: 29.97 fps

- no ear tracking

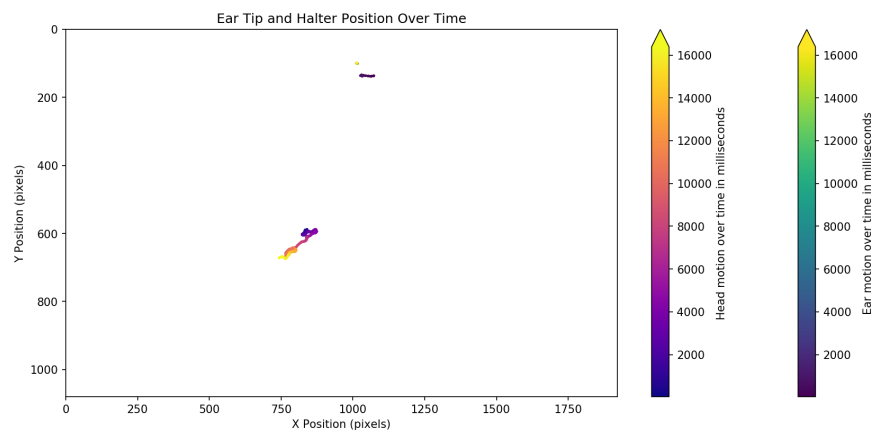


Figure B.39: 2D Head and ear motion with color changing with respect to time.

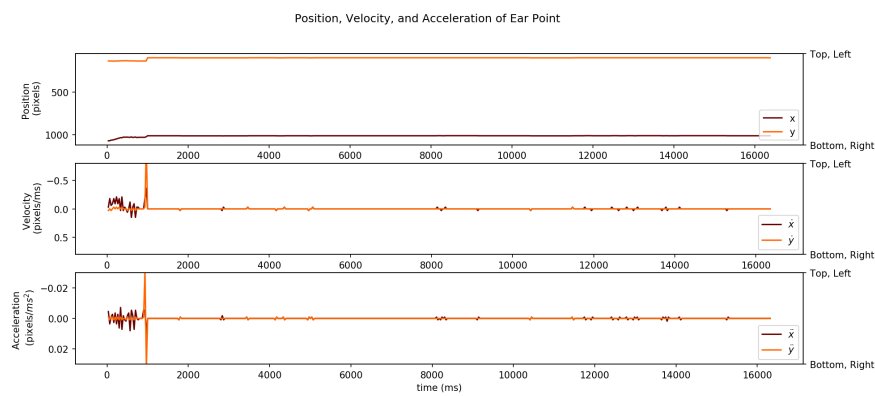


Figure B.40: Ear position, velocity and acceleration.

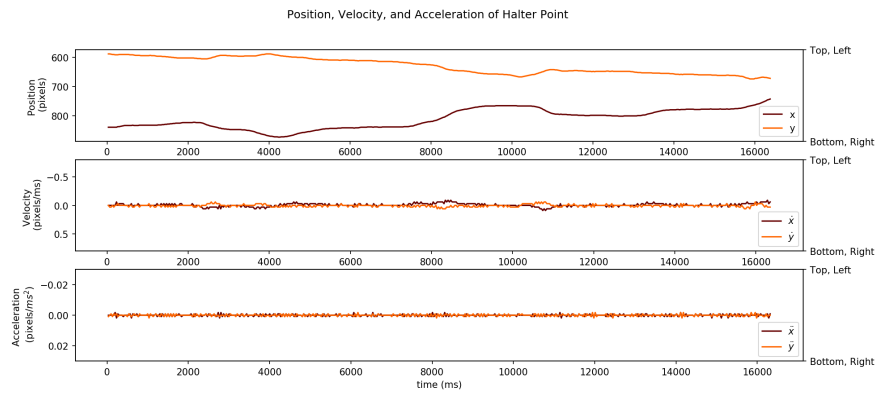


Figure B.41: Ear movement measurements.

B.14 mvi_4904.mov

Horse: Shaggy Pony
Frame size: 1920 x 1080 pixels
Frame rate: 29.97 fps

B.15 mvi_4905.mov

Horse: Shaggy Pony
Frame size: 1280 x 720 pixels
Frame rate: 59.94 fps

- ear moves forward
- ear moves side ways fr 318
- back forward fr 392
- back 416
- forwad 445
- not much head movement

B.16 mvi_4906.mov

Horse: The Dude
Frame size: 1280 x 720 pixels
Frame rate: 59.94 fps

B.17 mvi_4907.mov

Horse: The Dude
Frame size: 1920 x 1080 pixels
Frame rate: 29.97 fps

- head turns to look at camera
- a little ear movement
- head turns to look at camera no ear motion

B.18 mvi_4908.mov

Horse: Pirate

Frame size: 1920 x 1080 pixels

Frame rate: 29.97 fps

B.19 mvi_4909.mov

Horse: Pirate

Frame size: 1920 x 1080 pixels

Frame rate: 29.97 fps

Did not use, because horse is facing to the right.

B.20 mvi_4910.mov

Horse: Pirate

Frame size: 1280 x 720 pixels

Frame rate: 59.94 fps

Did not use, because horse is facing to the right.

B.21 mvi_4911.mov

Background video no horse.

B.22 mvi_4912.mov

Horse: Bradd Pit

Frame size: 1280 x 720 pixels

Frame rate: 59.94 fps

Did not use because handler was in frame.

B.23 mvi_4913.mov

Horse: Bradd pit

Frame size: 1920 x 1080 pixels

Frame rate: 29.97 fps

Did not use because handler was in frame.

B.24 mvi_4914.mov

Horse: Titanic

Frame size: 1920 x 1080 pixels

Frame rate: 29.97 fps

- no ear
- short
- looks at camera

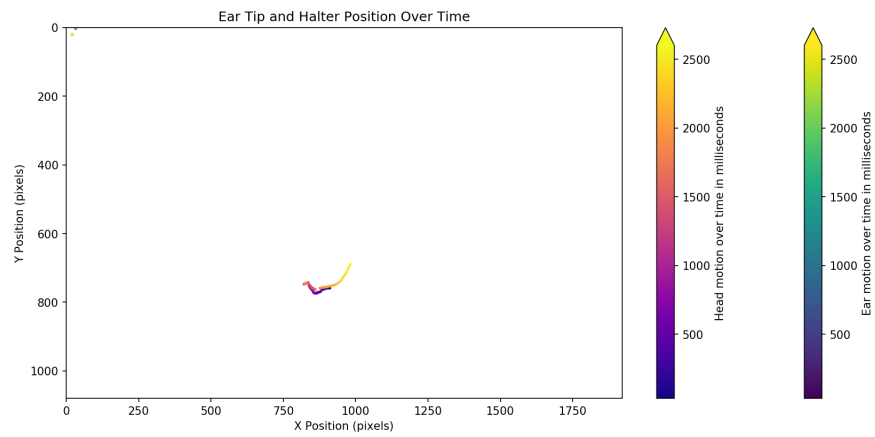


Figure B.42: 2D Head and ear motion with color changing with respect to time.

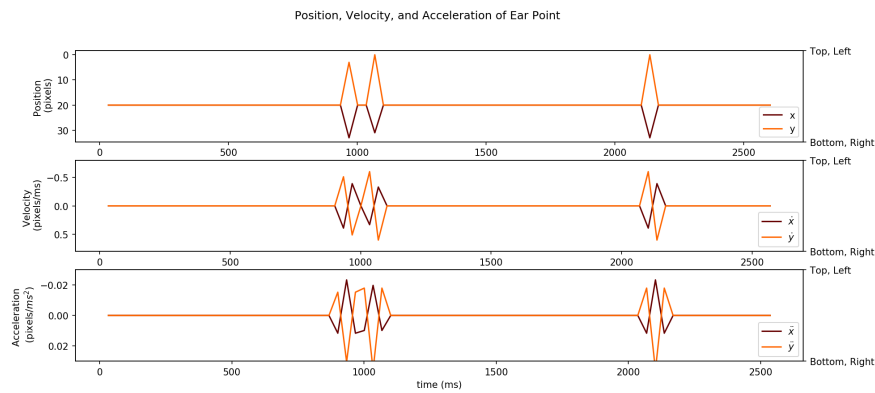


Figure B.43: Ear position, velocity and acceleration.

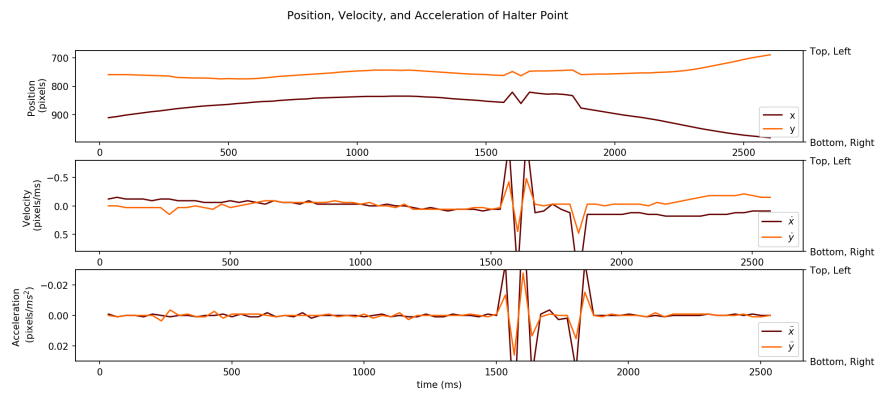


Figure B.44: Ear movement measurements.

B.25 mvi_4915.mov

Horse: Titanic

Frame size: 1280 x 720 pixels

Frame rate: 59.94 fps

- edit 01
- no ear
- purple blanket
- looks at camera fr 675-782
- looks away 810
- edit 02
- looking away from camera
- looks forward 381

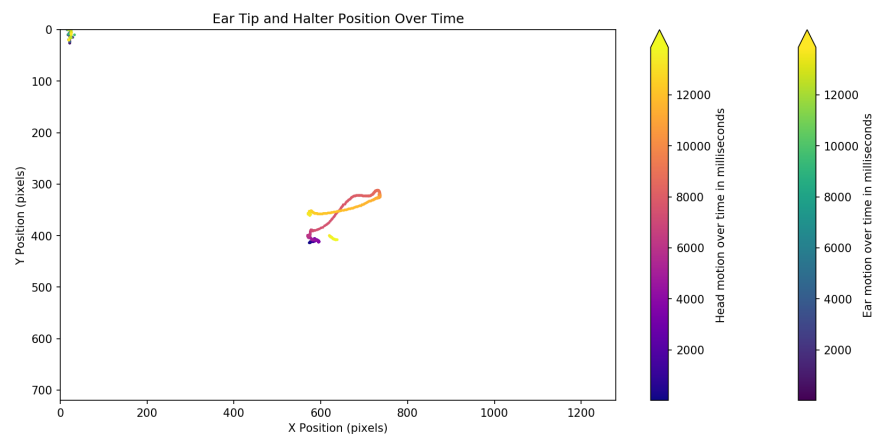


Figure B.45: 2D Head and ear motion with color changing with respect to time.

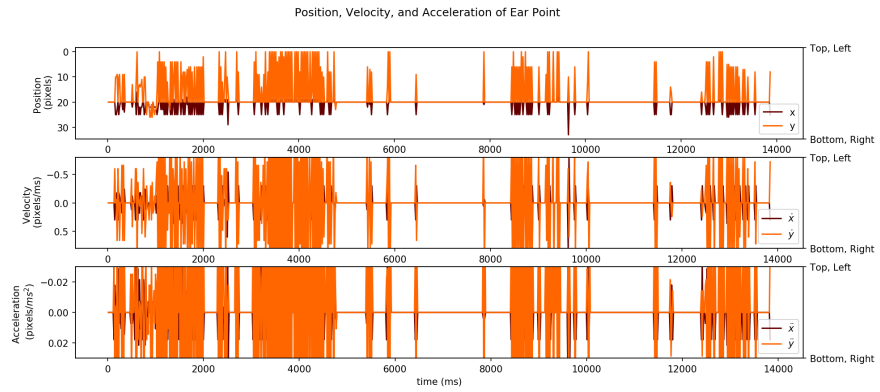


Figure B.46: Ear position, velocity and acceleration.

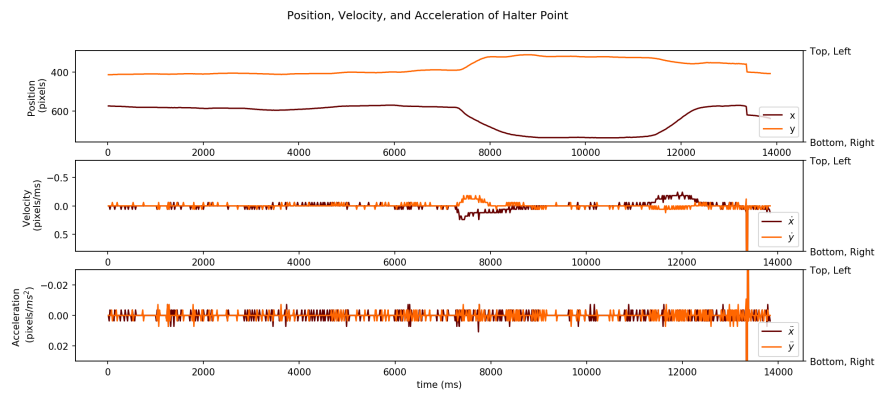


Figure B.47: Ear movement measurements.

B.26 mvi_4916.mov

Background video no horse.

B.27 mvi_4917.mov

Horse: Tethys

Frame size: 1280 x 720 pixels

Frame rate: 59.94 fps

- very little head motion
- a little ear motion
- good ear track

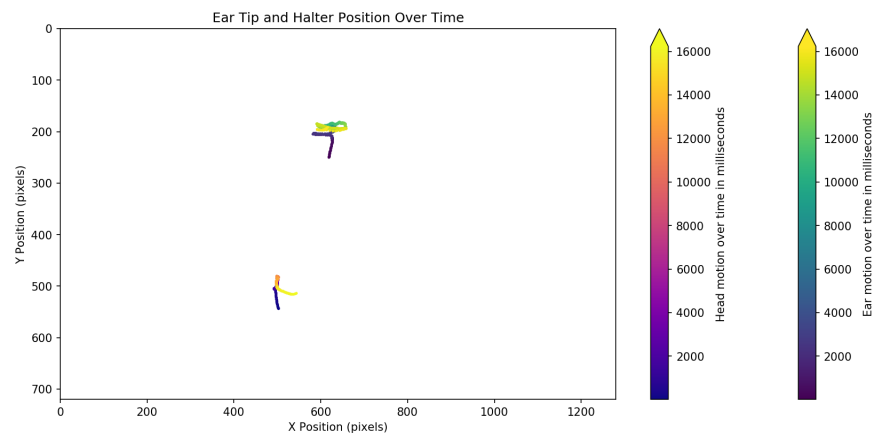


Figure B.48: 2D Head and ear motion with color changing with respect to time.

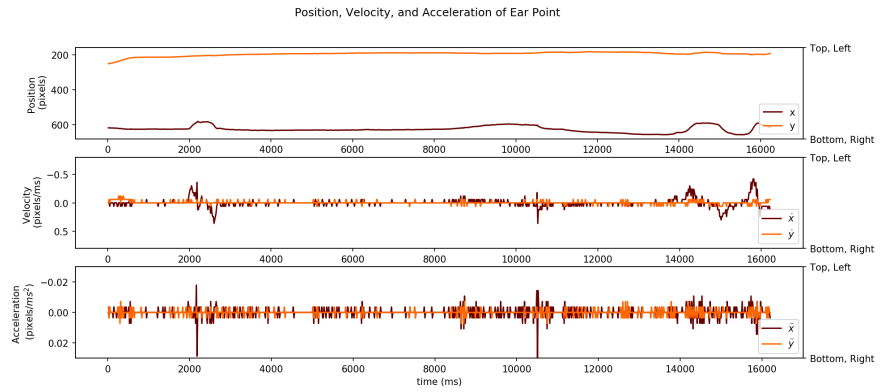


Figure B.49: Ear position, velocity and acceleration.

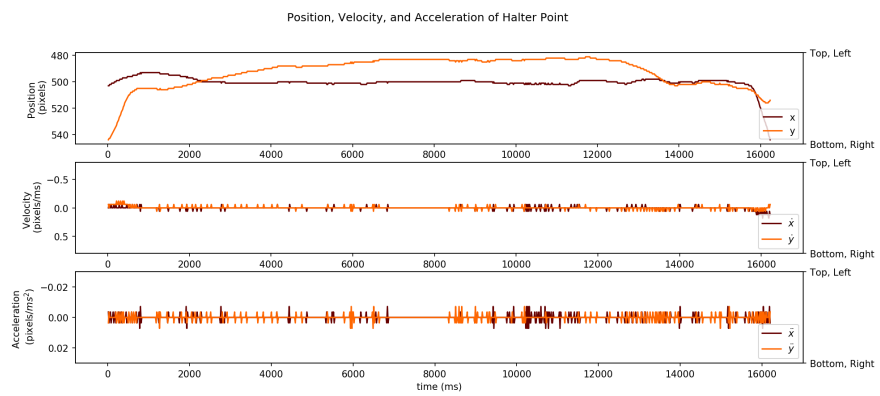


Figure B.50: Ear movement measurements.

B.28 mvi_4918.mov

Horse: Tethys

Frame size: 1920 x 1080 pixels

Frame rate: 29.97 fps

- edit 01
- little motion
- loses ear at end
- edit 02
- no ear track
- very little motion

B.29 mvi_4919.mov

Horse: Tethys

Frame size: 1920 x 1080 pixels

Frame rate: 29.97 fps

Appendix C

Equipment and Code

C.1 Equipment

- Canon 60D
- 18-55mm
- Lenovo Yoga 2 11 with an Intel Pentium processor

C.2 Software

- Linux Mint 18.1
- Python 3.5.2
- OpenCV 3.2.0
- Numpy 1.12.1
- Matplotlib 2.0.2
- git for version control

due to frame rate differences between the original video and the output video the lengths of the two videos are different.

C.3 OpenCV Installation

Functions used in this thesis required OpenCV 3, but OpenCV 2 had already been installed on the computer. Installing OpenCV 3 after the fact can cause compatibility issues between OpenCV 2 and 3. The best way around this is to install OpenCV 3 in a python virtual environment. "Ubuntu 16.04: How to install OpenCV" [\[36\]](#) is the tutorial that was used to install OpenCV 3.

C.4 Source Code

C.4.1 horse_head_tacker.py

Listing C.1: Complete source code used for this thesis.

```
1 #!/usr/bin/env python3
2
3 import cv2
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from matplotlib import cm
7 import csv
8
9 VIDEO_FILE = '../..Videos/no sound/edited/4893_edit01_1920_1080.mov'
10 # light, dark horse color flag.
11 horse_color = 0
12
13 # Image filters.
14 def halter_filter(img):
15     """Runs HSV filter on image. Returns binary mask image of halter.
16     """
17
18     # HSV color filter values for purple halter.
19     h_low = 133
20     h_high = 160
21     s_low = 56
22     s_high = 182
23     v_low = 74
24     v_high = 255
25
26     # Blur the image to help remove aliasing.
27     blur = cv2.GaussianBlur(img,(11,11),0)
28
29     # kernel for eroding
30     kernel = np.ones((5,5),np.uint8)
31
32     hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
33
34     lower_hsv = np.array([h_low,s_low,v_low])
35     upper_hsv = np.array([h_high,s_high,v_high])
36
37     mask = cv2.inRange(hsv, lower_hsv, upper_hsv)
```

```

38     mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel) # erode then
        dialte
39
40     return mask
41
42 def canny_edge(img):
43     """Returns the canny edge after converting image to grayscale"""
44     # gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
45
46     blur = cv2.GaussianBlur(img,(7,7),0)
47
48     edges = cv2.Canny(blur,60,100)
49
50     return edges
51
52 def background_subtraction_mask(img, halter_point, firstrun = False):
53     '''Returns a mask of just the horse body.'''
54     # Bring global flag.
55     global horse_color
56
57     # Split the x and y point.
58     halt_x = halter_point[0]
59     halt_y = halter_point[1]
60
61     # Convert image to HSV.
62     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
63
64     # Set the horse color light or dark in the firstrun
65     if(hsv[halt_y, halt_x-20][2] > 50 and firstrun == True):
66         horse_color = 0
67     elif(hsv[halt_y, halt_x-20][2] <= 50 and firstrun == True):
68         horse_color = 1
69     else:
70         None
71
72     if(horse_color == 0):
73         # HSV color filter values for light horse
74         h_low = 0
75         h_high = 25
76         s_low = 64
77         s_high = 184
78         v_low = 83
79         v_high = 255
80
81     else:

```



```

82     # HSV color filter values for dark horse
83     h_low = 0
84     h_high = 255
85     s_low = 0
86     s_high = 255
87     v_low = 0
88     v_high = 50
89
90     lower_hsv = np.array([h_low, s_low, v_low])
91     upper_hsv = np.array([h_high, s_high, v_high])
92
93     mask = cv2.inRange(hsv, lower_hsv, upper_hsv)
94
95     # kernel for eroding
96     kernel = np.ones((5,5), np.uint8)
97     mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel) # erode then
98     dialte
99
100     return mask
101
102 # Point tracking.
103 def find_top_halter_point(img):
104     """Returns the top point of a binary image."""
105     if (img.max() != 0):
106         row, col = np.where(img==255)
107         return col[0], row[0]
108     else:
109         return 0, 0
110
111 def find_ear_tip(halter_point, canny_img):
112     """Finds the ear tip from the top of the halter. Used to initially
113     find the
114     ear tip."""
115     finding_ear = 1 # Flag for loop to find ear tip
116     moving_left = 1 # Flag to prevent constantly moving back and
117     forth to find ear tip.
118
119     # Setup bounds so if the ear point goes above them it is a bad
120     point.
121     # Gets the image height and divides by 4 because the ear point
122     # should never be higher than a quarter frame above the halter point
123
124     row, col = canny_img.shape
125     bounds = row/4.0

```

```

122     halt_x = halter_point[0]
123     halt_y = halter_point[1]
124
125     # print("canny")
126     # print(canny_img[0:halt_y, halt_x])
127
128     # Look line above the top halter point.
129     if(canny_img[0:halt_y, halt_x].max() != 0):
130         point_y = np.where(canny_img[0:halt_y, halt_x] == 255)[0][0]
131         ear_tip = halt_x, point_y
132
133     # If there is no line above the top halter point, move left to find
134     # line.
135     elif(canny_img[halt_y, 0:halt_x].max() != 0):
136         # store the row so it can be reversed.
137         row = canny_img[halt_y, 0:halt_x]
138         # reverse the row because to look from right to left to find
139         # line.
140         row = row[::-1]
141         # Find the line. The first place where 255 shows up.
142         diff_x = np.where(row == 255)[0][0]
143         # Subtract the point in the small row from halt_x to get the
144         # location
145         # of the line in the whole image.
146         point_x = halt_x - diff_x
147         # Look for a line above the point just found.
148         point_y = np.where(canny_img[0:halt_y, point_x] == 255)[0][0]
149
150         ear_tip = point_x, point_y
151
152     else:
153         ear_tip = 20, 20
154         finding_ear = 0
155
156     while(finding_ear):
157         # If top left point = 255 and moving left.
158         if(canny_img[ear_tip[1]-1, ear_tip[0]-1] == 255 and moving_left
159            == 1):
160             ear_tip = ear_tip[0]-1, ear_tip[1]-1
161             finding_ear = 1
162
163         # If top point = 255 and moving left.
164         elif(canny_img[ear_tip[1]-1, ear_tip[0]] == 255 and moving_left
165            == 1):
166             ear_tip = ear_tip[0], ear_tip[1]-1

```

```
162         finding_ear = 1
163
164     # If left point = 255 and moving left.
165     elif(canny_img[ear_tip[1], ear_tip[0]-1] == 255 and moving_left
166          == 1):
167         ear_tip = ear_tip[0]-1, ear_tip[1]
168         finding_ear = 1
169
170     # If bottom left point = 255 and moving left.
171     elif(canny_img[ear_tip[1]+1, ear_tip[0]-1] == 255 and
172          moving_left == 1):
173         moving_left = 0
174
175     # If bottom point = 255 and moving left.
176     elif(canny_img[ear_tip[1]+1, ear_tip[0]] == 255 and moving_left
177          == 1):
178         moving_left = 0
179
180     # If top right point = 255 and moving right.
181     elif(canny_img[ear_tip[1]-1, ear_tip[0]+1] == 255 and
182          moving_left == 0):
183         ear_tip = ear_tip[0]+1, ear_tip[1]-1
184         finding_ear = 1
185
186     # If top point = 255 and moving right.
187     elif(canny_img[ear_tip[1]-1, ear_tip[0]] == 255 and moving_left
188          == 0):
189         ear_tip = ear_tip[0], ear_tip[1]-1
190         finding_ear = 1
191
192     # If right point = 255 and moving right.
193     elif(canny_img[ear_tip[1], ear_tip[0]+1] == 255 and moving_left
194          == 0):
195         ear_tip = ear_tip[0]+1, ear_tip[1]
196         finding_ear = 1
197
198     else:
199         ear_tip = ear_tip[0], ear_tip[1]
200         finding_ear = 0
201
202     return ear_tip
203
204 def ear_tracker(ear_tip_point, canny_img):
205     '''Used to track the ear after the tip has been found.'''
206     finding_ear = 1 # Flag for loop to find ear tip
```

```

201     moving_left = 1      # Flag to prevent constantly moving back and
                          # forth to find ear tip.
202
203     # Setup bounds so if the ear point goes above them it is a bad
                          # point.
204     # Gets the image height and divides by 4 because the ear point
205     # should never be higher than a quarter frame above the halter point
                          .
206     row, col = canny_img.shape
207     bounds = row/4.0
208
209     ear_x, ear_y = ear_tip_point
210     # print(canny_img[ear_y-10:ear_y+10, ear_x-10:ear_x+10])
211     point = np.where(canny_img[ear_y-10:ear_y+10, ear_x-10:ear_x+10] ==
212                     255)
213     # print(point)
214     # print(len(point[0]))
215
216     if(len(point[0]) == 0):
217         return -1,-1
218     else:
219         ear_tip_x = ear_x - 10 + point[1][0]
220         ear_tip_y = ear_y - 10 + point[0][0]
221         ear_tip = ear_tip_x, ear_tip_y
222         # print(canny_img[ear_tip_y-10:ear_tip_y+10, ear_tip_x-10:
223             ear_tip_x+10])
224
225     while(finding_ear):
226         # If top left point = 255 and moving left.
227         if(canny_img[ear_tip[1]-1, ear_tip[0]-1] == 255 and
228             moving_left == 1):
229             ear_tip = ear_tip[0]-1, ear_tip[1]-1
230             finding_ear = 1
231
232         # If top point = 255 and moving left.
233         elif(canny_img[ear_tip[1]-1, ear_tip[0]] == 255 and
234             moving_left == 1):
235             ear_tip = ear_tip[0], ear_tip[1]-1
236             finding_ear = 1
237
238         # If left point = 255 and moving left.
239         elif(canny_img[ear_tip[1], ear_tip[0]-1] == 255 and
240             moving_left == 1):
241             ear_tip = ear_tip[0]-1, ear_tip[1]
242             finding_ear = 1

```

```

238
239     # If bottom left point = 255 and moving left.
240     elif(canny_img[ear_tip[1]+1, ear_tip[0]-1] == 255 and
241           moving_left == 1):
242         moving_left = 0
243
244     # If bottom point = 255 and moving left.
245     elif(canny_img[ear_tip[1]+1, ear_tip[0]] == 255 and
246           moving_left == 1):
247         moving_left = 0
248
249     # If top right point = 255 and moving right.
250     elif(canny_img[ear_tip[1]-1, ear_tip[0]+1] == 255 and
251           moving_left == 0):
252         ear_tip = ear_tip[0]+1, ear_tip[1]-1
253         finding_ear = 1
254
255     # If top point = 255 and moving right.
256     elif(canny_img[ear_tip[1]-1, ear_tip[0]] == 255 and
257           moving_left == 0):
258         ear_tip = ear_tip[0], ear_tip[1]-1
259         finding_ear = 1
260
261     # If right point = 255 and moving right.
262     elif(canny_img[ear_tip[1], ear_tip[0]+1] == 255 and
263           moving_left == 0):
264         ear_tip = ear_tip[0]+1, ear_tip[1]
265         finding_ear = 1
266
267     else:
268         ear_tip = ear_tip[0], ear_tip[1]
269         finding_ear = 0
270
271     return ear_tip
272
273     # if(canny_img[0:halt_y, halt_x].max() != 0):
274     #     point_y = np.where(canny_img[0:halt_y, halt_x] == 255)[0][0]
275     #     ear_tip = halt_x, point_y
276     # else:
277     #     ear_tip = 0, 0
278     #
279
280 def eye_tracker(img, no_bg_img):
281

```

```
278 # Setup SimpleBlobDetector parameters.
279 params = cv2.SimpleBlobDetector_Params()
280
281 # Change thresholds
282 # params.minThreshold = 0;
283 # params.maxThreshold = 200;
284
285 # Filter by color
286 params.filterByColor = True
287 params.blobColor = 0
288
289 # Filter by Area.
290 params.filterByArea = True
291 params.minArea = 250
292 # params.maxArea = 5000
293
294 # Filter by Circularity
295 params.filterByCircularity = False
296 params.minCircularity = 0.1
297
298 # Filter by Convexity
299 params.filterByConvexity = False
300 params.minConvexity = 0.87
301
302 # Filter by Inertia
303 params.filterByInertia = False
304 params.minInertiaRatio = 0.01
305
306 # Set up the detector with default parameters.
307 detector = cv2.SimpleBlobDetector_create(params)
308
309 # Detect blobs.
310 keypoints = detector.detect(no_bg_img)
311 # print(keypoints)
312
313 # Draw detected blobs as red circles.
314 # cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures the size of
   the circle corresponds to the size of blob
315 im_with_keypoints = cv2.drawKeypoints(no_bg_img, keypoints, np.
   array([]), (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
   )
316
317 # cv2.imshow('image', mask)
318 # img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY )
319 # img = cv2.medianBlur(img, 5)
```

```

320 # cimg = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
321 # circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT, 1, 20,
322 #                             param1=50, param2=30, minRadius=0,
323 #                             maxRadius=0)
324 # circles = np.uint16(np.around(circles))
325 # for i in circles[0,:]:
326 #     # draw the outer circle
327 #     cv2.circle(cimg, (i[0], i[1]), i[2], (0, 255, 0), 2)
328 #     # draw the center of the circle
329 #     cv2.circle(cimg, (i[0], i[1]), 2, (0, 0, 255), 3)
330
331 # return eye_x, eye_y
332 return im_with_keypoints
333
334 # Head tracking.
335 def find_halter_point(halter_mask, img):
336     ''' Finds halter points to track from the mask of the halter. '''
337     rhos = []
338     angles = []
339     ave_lines = []
340     line_points = []
341
342     edges = cv2.Canny(halter_mask, 50, 150, apertureSize = 3)
343
344     lines = cv2.HoughLines(edges, 1, np.pi/180, 50)
345
346     if lines != None:
347         lines = lines[:, 0] # reshape array from (23, 1, 2) to (23, 2)
348         lines_sorted = lines[lines[:, 1].argsort()] # Sort by column 2
349         # where the angles are stored
350
351         rho, prev_theta = lines_sorted[0]
352         for rho, theta in lines_sorted:
353             diff = theta - prev_theta
354             # print("Diff = {}, theta = {}, rho = {}".format(diff, theta
355             # , rho))
356             if diff <= 0.1:
357                 angles.append(theta)
358                 rhos.append(rho)
359
360             elif diff > 0.1:
361                 # print("distance greater than 0.1")
362                 # print("Angles: {}".format(angles))
363                 ave_lines.append([sum(rhos)/len(rhos), sum(angles)/len(
364                     angles)])

```

```

361         angles = []
362         rhos = []
363         angles.append(theta)
364         rhos.append(rho)
365         dist = 0
366
367     else:
368         print("Error:")
369
370     prev_theta = theta
371
372     ave_lines.append([sum(rhos)/len(rhos), sum(angles)/len(angles)
373                    ])
374     angles = []
375     rhos = []
376
377     # Draw the Hough lines if there are any in red.
378     if lines != None:
379         for rho, theta in lines:
380             a = np.cos(theta)
381             b = np.sin(theta)
382             x0 = a*rho
383             y0 = b*rho
384             x1 = int(x0 + 1000*(-b))
385             y1 = int(y0 + 1000*(a))
386             x2 = int(x0 - 1000*(-b))
387             y2 = int(y0 - 1000*(a))
388
389             cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 1)
390
391     lines = []
392
393     # Calculate the points for the average lines in blue.
394     for rho, theta in ave_lines:
395         a = np.cos(theta)
396         b = np.sin(theta)
397         x0 = a*rho
398         y0 = b*rho
399         x1 = int(x0 + 1000*(-b))
400         y1 = int(y0 + 1000*(a))
401         x2 = int(x0 - 1000*(-b))
402         y2 = int(y0 - 1000*(a))
403
404     # Draw the average lines.
405     cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 2)

```



```

405     line_points.append((x1,y1,x2,y2))
406     # head_angle.append(find_angle(ave_lines[1][1]))
407
408     ave_lines = []
409
410     int_x, int_y = find_intersection(line_points)
411
412     return int_x, int_y
413
414 def find_intersection(points):
415
416     # print("Nuber of lines: {}".format(len(points)))
417     # print("Line points: {}".format(points))
418     if len(points) > 1:
419         x1_0,y1_0,x2_0,y2_0 = points[0]
420         x1_1,y1_1,x2_1,y2_1 = points[1]
421
422         # checks for a vertical line.
423         # vertical lines cause a divide by zero error because there is
424         # no change in x.
425         # so the the angle has to be manually set.
426         if x2_0 - x1_0 == 0:
427             m0 = 90 / 180 * np.pi          # Manually set vertical angle
428             in radians
429         else:
430             m0 = (y2_0 - y1_0) / (x2_0 - x1_0)
431
432         if x2_1 - x1_1 == 0:
433             m1 = 90 / 180 * np.pi
434         else:
435             m1 = (y2_1 - y1_1) / (x2_1 - x1_1)
436
437         # print(m0, m1)
438
439         a = np.array([[ -m0, 1 ], [ -m1, 1 ]])
440         b = np.array([[ y1_0 - (m0*x1_0) ], [ y1_1 - (m1*x1_1) ]])
441         p_x,p_y = np.linalg.solve(a,b)
442
443         # print(p_x,p_y)
444     else:
445         print("Not enough lines.")
446         p_x,p_y = 0,0
447
448     # TODO finish following code

```

```

448 # lines below are unfinished and are to make the code independent
      of the number of lines
449 # if len(points) > 1:
450 #     for index in range(0, len(points)-1 ):
451 #         x1_0,y1_0,x2_0,y2_0 = points[index]
452 #         x1_1,y1_1,x2_1,y2_1 = points[index+1]
453 #
454 #         m1 =
455 #
456 # else:
457 #     print("Not enough lines.")
458
459 return p_x, p_y
460
461 def halter_contour(halter_mask):
462     '''Finds the halter using countours.'''
463     im2, contours, hierarchy = cv2.findContours(halter_mask, cv2.
        RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
464
465     # cnt_filtered = []
466     bounding_rect = []
467     bounding_box_points = []
468     angles = []
469     lr_sorted = []
470     # cv2.drawContours(frame, contours, -1, (0,255,0), 3)
471
472     for cnt in contours:
473         area = cv2.contourArea(cnt)
474         # print("Area: {}".format(area))
475         if(area <= 400):
476             None
477         else:
478             rect = cv2.minAreaRect(cnt)
479             # print("Angle: {}".format(rect))
480             box = cv2.boxPoints(rect)
481             box = np.int0(box)
482             # print(box)
483             # cv2.drawContours(frame, [box], 0, (0,255,0), 2)
484             # cnt_filtered.append(cnt)
485             bounding_box_points.append(box)
486             bounding_rect.append(rect)
487             # Find the angle based on long edge of rectangle.
488             if(rect[1][1] > rect[1][0]):
489                 angles.append(rect[2] + 90)
490             else:

```

```

491         angles.append(rect[2])
492
493     # Reverse the order of contours and bounding boxes
494     # cnt_filtered = cnt_filtered[::-1]
495     bounding_box_points = bounding_box_points[::-1]
496     bounding_rect = bounding_rect[::-1]
497     angles = angles[::-1]
498     # print(angles)
499     # print(np.diff(angles))
500
501     top_box = bounding_rect[0]
502     # print(top_box)
503     high_point = (int(top_box[0][0]), int(top_box[0][1]))
504     # print(high_point)
505
506     # mid_box = bounding_rect[2]
507     # mid_point = (mid_box[0][0], mid_box[0][1])
508     # print(mid_point)
509
510     # Sort bounding rectangles left to right using center point.
511     print("Sorting...")
512     lr_sorted = sorted(bounding_rect, key=lambda rect_point: rect_point
513                        [0][0])
514     left_box = lr_sorted[0]
515     left_point = (int(left_box[0][0]), int(left_box[0][1]))
516
517     return high_point, left_point, bounding_box_points
518
519 # Drawing.
520 def overlay_binary_image(img, binary):
521     """Overlays a binary image in red onto original image."""
522
523     red_binary = np.zeros(img.shape, dtype=np.uint8)
524     # Create blank frame the size of the binary image.
525     # row, col = binary.shape
526     # blank = np.zeros((row, col, 1))
527
528     # Turn binary image to color image with the binary image on the red
529     # channel.
530     # red_binary = cv2.merge((blank, blank, binary))
531     red_binary[:, :, 2] = binary
532
533     overlay = cv2.add(img, red_binary)
534
535     return overlay

```

```

534
535 def derivative(a,b):
536     ''' takes the derivative of matrix a with respect to matrix b '''
537     da = np.diff(a, axis=0)
538     db = np.diff(b, axis=0)
539
540     # print("A ", len(a))
541     # print("B ", len(b))
542
543     # print(np.divide(da, db))
544     # print(da/db)
545
546     # return da/db
547     return np.divide(da, db)
548
549 def graph_data(frame_size, time, ear_points, halter_track):
550     '''Graphs all of the data.'''
551
552     # Restructure the data to be graphed.
553     ear_x = []
554     ear_y = []
555     for element in ear_points:
556         ear_x.append(element[0])
557         ear_y.append(element[1])
558
559     halt_x = []
560     halt_y = []
561     for element in halter_track:
562         halt_x.append(element[0])
563         halt_y.append(element[1])
564
565     # Calculate the first derivative
566     ear_x_dot = derivative(ear_x,time)
567     ear_y_dot = derivative(ear_y,time)
568     halt_x_dot = derivative(halt_x,time)
569     halt_y_dot = derivative(halt_y,time)
570
571     # Calculate the second derivative
572     ear_x_ddot = derivative(ear_x_dot,time[0:len(time)-1])
573     ear_y_ddot = derivative(ear_y_dot,time[0:len(time)-1])
574     halt_x_ddot = derivative(halt_x_dot,time[0:len(time)-1])
575     halt_y_ddot = derivative(halt_y_dot,time[0:len(time)-1])
576
577     ## Scatter plot.
578     plt.figure()

```

```

579 plt.title("Ear Tip and Halter Position Over Time")
580 plt.axis([0, frame_size[0], 0, frame_size[1]])
581 plt.xlabel("X Position (pixels)")
582 plt.ylabel("Y Position (pixels)")
583 plt.scatter(ear_x, ear_y, 2, c=time, cmap=cm.viridis)
584 cb = plt.colorbar()
585 cb.set_label('Ear motion over time in milliseconds')
586 plt.scatter(halt_x, halt_y, 2, c=time, cmap=cm.plasma)
587 plt.gca().invert_yaxis()
588 cb = plt.colorbar()
589 cb.set_label('Head motion over time in milliseconds')
590
591 ## Ear plot.
592 #####
593 fig, axes = plt.subplots(3,1)
594 fig.suptitle("Position and Motion of Ear Point")
595 axes[0].plot(time, ear_x, label='x', color="#660000")
596 axes[0].plot(time, ear_y, label='y', color="#FF6600")
597 axes[0].set_ylabel('(pixels)')
598 axes[0].legend(loc='lower right')
599 # Plot the first derivative, velocity.
600 axes[1].plot(time[0:len(time)-1], ear_x_dot, label=r'$\dot{x}$',
601             color="#660000")
602 axes[1].plot(time[0:len(time)-1], ear_y_dot, label=r'$\dot{y}$',
603             color="#FF6600")
604 axes[1].set_ylabel('(pixels/ms)')
605 axes[1].legend(loc='lower right')
606 # Plot the second derivative, acceleration.
607 axes[2].plot(time[0:len(time)-2], ear_x_ddot, label=r'$\ddot{x}$',
608             color="#660000")
609 axes[2].plot(time[0:len(time)-2], ear_y_ddot, label=r'$\ddot{y}$',
610             color="#FF6600")
611 axes[2].set_ylabel(r'(pixels/$ms^2$)')
612 axes[2].legend(loc='lower right')
613 plt.xlabel('time (ms)')
614
615 ## Halter point.
616 #####
617 fig, axes = plt.subplots(3,1)
618 fig.suptitle("Position and Motion of Halter Point")
619 axes[0].plot(time, halt_x, label='x', color="#660000")
620 axes[0].plot(time, halt_y, label='y', color="#FF6600")
621 axes[0].set_ylabel('(pixels)')
622 axes[0].legend(loc='lower right')
623 # Plot the first derivative, velocity.

```

```

618 axes[1].plot(time[0:len(time)-1], halt_x_dot, label=r'\dot{x}$',
619             color="#660000")
619 axes[1].plot(time[0:len(time)-1], halt_y_dot, label=r'\dot{y}$',
620             color="#FF6600")
620 axes[1].set_ylabel('(pixels/ms)')
621 axes[1].legend(loc='lower right')
622 # Plot the second derivative, acceleration.
623 axes[2].plot(time[0:len(time)-2], halt_x_ddot, label=r'\ddot{x}$',
624             color="#660000")
624 axes[2].plot(time[0:len(time)-2], halt_y_ddot, label=r'\ddot{y}$',
625             color="#FF6600")
625 axes[2].set_ylabel(r'(pixels/$ms^2$)')
626 axes[2].legend(loc='lower right')
627 plt.xlabel('time (ms)')
628
629 def csv_output(fieldnames, time, frame_num, ear_point, halter_point):
630     ear_x = ear_point[0]
631     ear_y = ear_point[1]
632
633     halt_x = halter_point[0]
634     halt_y = halter_point[1]
635
636     # fieldnames = ['Time', 'Frame Number', 'Ear X', 'Ear Y', 'Halter X
637     #               ', 'Halter Y']
637     with open('%s.csv' %VIDEO_FILE, 'a') as csvfile:
638         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
639         writer.writerow({'Time': time, 'Frame Number': frame_num, 'Ear
640                         X': ear_x, 'Ear Y': ear_y, 'Halter X': halt_x, 'Halter Y':
641                         halt_y})
642
643 def main():
644     cap = cv2.VideoCapture(VIDEO_FILE)
645
646     # Font for writing on frames.
647     font = cv2.FONT_HERSHEY_SIMPLEX
648
649     # Open csv file.
650     fieldnames = ['Time', 'Frame Number', 'Ear X', 'Ear Y', 'Halter X',
651                 'Halter Y']
652     with open('%s.csv' %VIDEO_FILE, 'w') as csvfile:
653         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
654         writer.writeheader()
655
656     frame_width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
657     frame_height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

```

```

655     frame_size = frame_width, frame_height
656     fps = cap.get(cv2.CAP_PROP_FPS)
657
658     # Define the codec and create VideoWriter object
659     fourcc = cv2.VideoWriter_fourcc(*'XVID')
660     out = cv2.VideoWriter('%s_output.avi' %VIDEO_FILE, fourcc, 15.0, (
661         int(frame_width),int(frame_height))
662     out2 = cv2.VideoWriter('%s_halt_mask.avi' %VIDEO_FILE, fourcc,
663         15.0, (int(frame_width),int(frame_height))
664     out3 = cv2.VideoWriter('%s_head_mask.avi' %VIDEO_FILE, fourcc,
665         15.0, (int(frame_width),int(frame_height))
666     out4 = cv2.VideoWriter('%s_edges.avi' %VIDEO_FILE, fourcc, 15.0, (
667         int(frame_width),int(frame_height))
668
669     # Automatically choose the scale factor to scale videos before
670     displaying.
671     if(frame_height == 1080):
672         scale_factor = 0.6
673     else:
674         scale_factor = 0.8
675
676     frame_num = 0
677     firstrun = True
678
679     cv2.namedWindow('Video', cv2.WINDOW_NORMAL)
680     # cv2.namedWindow('Halter Mask', cv2.WINDOW_NORMAL)
681     # cv2.namedWindow('Canny Edges', cv2.WINDOW_NORMAL)
682
683     # lists for graphing
684     time = []
685     ear_track = []
686     halter_track = []
687
688     while(cap.isOpened()):
689         ret, frame = cap.read()
690
691         if ret == False:
692             print("End of video.")
693             break
694
695         else:
696             print("Frame number: {}".format(frame_num))
697             # Altered images.
698             halter_mask = halter_filter(frame)

```

```

695     ## Halter Traking
696         #####
697     # hatler_intersection = find_halter_point(halter_mask,
698         frame)
699     top_halter_point, mid_halter_point, bounding_boxes =
700         halter_contour(halter_mask)
701     # print("Top halter point: {}".format(top_halter_point[0]))
702
703     no_bg_mask = background_subtraction_mask(frame,
704         top_halter_point, firstrun)
705     edges = canny_edge(no_bg_mask)
706
707     # Image of color masked horse body.
708     horse_body = cv2.bitwise_and(frame, frame, mask = no_bg_mask)
709
710     ## Ear tracking.
711         #####
712     # top_halter_point = find_top_halter_point(halter_mask)
713     if (firstrun):
714         ear_point = find_ear_tip(top_halter_point, edges)
715     else:
716         if (ear_point[0] == 0 and ear_point[1] == 0):
717             ear_point = find_ear_tip(top_halter_point, edges)
718         else:
719             ear_point = ear_tracker(ear_point, edges)
720             if (ear_point[0] == -1):
721                 ear_point = find_ear_tip(top_halter_point,
722                     edges)
723
724     ## Eye Tracking
725         #####
726     eye_frame = eye_tracker(frame, no_bg_mask)
727
728     ## Draw items
729         #####
730     cv2.circle(frame, top_halter_point, 5, (0,255,0), -1)
731     cv2.circle(frame, ear_point, 5, (255,0,0), -1)
732     cv2.circle(frame, mid_halter_point, 5, (255,0,0), -1)
733
734     for box in bounding_boxes:
735         cv2.drawContours(frame, [box], 0, (0,255,0), 2)
736     # cv2.drawContours(frame, [bounding_boxes[2]], 0, (255,0,0), 2)
737
738     final = overlay_binary_image(frame, edges)
739
740
741

```



```

732     cv2.rectangle(final , (5,5) , (250,35) , (0,0,0) ,-1)
733     cv2.putText(final , 'Frame: %d' %frame_num,(10,30) , font ,
734         1,(255,255,255) ,1,cv2.LINE_AA)
735     cv2.putText(halter_mask , 'Frame: %d' %frame_num,(10,30) ,
736         font , 1,(255,255,255) ,1,cv2.LINE_AA)
737     cv2.putText(no_bg_mask , 'Frame: %d' %frame_num,(10,30) , font
738         , 1,(255,255,255) ,1,cv2.LINE_AA)
739     halter_mask = cv2.cvtColor(halter_mask , cv2.COLOR_GRAY2BGR)
740     no_bg_mask = cv2.cvtColor(no_bg_mask , cv2.COLOR_GRAY2BGR)
741     edges = cv2.cvtColor(edges , cv2.COLOR_GRAY2BGR)
742
743     # Write frame to video.
744     out.write(final)
745     out2.write(halter_mask)
746     out3.write(no_bg_mask)
747     out4.write(edges)
748
749     ## Image displaying ##
750     scaled_final = cv2.resize(final ,None,fx=scale_factor , fy=
751         scale_factor ,
752         interpolation = cv2.INTER_CUBIC)
753     # scaled_mask = cv2.resize(halter_mask ,None,fx=scale_factor
754         , fy=scale_factor ,
755         # interpolation = cv2.INTER_CUBIC)
756     # scaled_edges = cv2.resize(edges ,None,fx=scale_factor , fy=
757         scale_factor ,
758         # interpolation = cv2.INTER_CUBIC)
759     # scaled_eye = cv2.resize(im_with_keypoints ,None,fx=
760         scale_factor , fy=scale_factor , interpolation = cv2.
761         INTER_CUBIC)
762
763     ## Data for graphing.
764     #####
765     time.append(cap.get(cv2.CAP_PROP_POS_MSEC))
766     ear_track.append(ear_point)
767     halter_track.append(mid_halter_point)
768
769     ## csv
770     csv_output(fieldnames ,cap.get(cv2.CAP_PROP_POS_MSEC) ,
771         frame_num ,ear_point ,mid_halter_point)
772
773     # eyes = eye_tracker(frame)
774
775     cv2.imshow('Video' ,scaled_final)
776     # cv2.imshow('Halter Mask' ,scaled_mask)

```

```

767         # cv2.imshow('Canny Edges', edges)
768
769         # Set first run to false because first program run is
           finished.
770         firstrun = False
771
772         if cv2.waitKey(1) & 0xFF == ord('q'):
773             print("Exiting")
774             break
775
776         frame_num += 1
777
778         # Graph everything.
779         graph_data(frame_size, time, ear_track, halter_track)
780
781         cv2.destroyAllWindows()
782         cap.release()
783         out.release()
784         out2.release()
785         out3.release()
786         out4.release()
787         plt.show()
788
789 if __name__ == '__main__':
790     main()

```

C.4.2 csv_to_graph.py

Listing C.2: Complete source code used for this thesis.

```

1 import csv
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib import cm
5 import matplotlib.patches as patches
6 import matplotlib.transforms as transforms
7 from scipy import signal
8 from scipy.fftpack import fft
9
10 # File name, locaiton and extension.
11 # makes changing file name and automatic figure saving easier.
12 folder_location = './18_11_04/'
13 csv_file = '4905_edit00_1280_720'
14 extension = '.mov.csv'

```

```

15 size = (1280, 720)
16
17 def derivative(a,b):
18     ''' takes the derivative of matrix a with respect to matrix b '''
19     da = np.diff(a, axis=0)
20     db = np.diff(b, axis=0)
21
22     return np.divide(da, db)
23
24 def graph_data(time, ear_x, ear_y, halt_x, halt_y, frame_size = (1920,
25     1080)):
26     '''Graphs all of the data.'''
27
28     # Filter the data lowpass 10 Hz
29     nyq = 0.5*30
30     normal_cutoff = 0.5 / nyq
31     b,a = signal.butter(1,normal_cutoff ,btype='lowpass')
32
33     # ear_x_filt = ear_x
34     # ear_y_filt = ear_y
35     # halt_x_filt = halt_x
36     # halt_y_filt = halt_y
37
38     ear_x_filt = signal.lfilter(b,a,ear_x)
39     ear_y_filt = signal.lfilter(b,a,ear_y)
40     halt_x_filt = signal.lfilter(b,a,halt_x)
41     halt_y_filt = signal.lfilter(b,a,halt_y)
42
43     # Calculate the first derivative
44     ear_x_dot = derivative(ear_x_filt ,time)
45     ear_y_dot = derivative(ear_y_filt ,time)
46     halt_x_dot = derivative(halt_x_filt ,time)
47     halt_y_dot = derivative(halt_y_filt ,time)
48
49     # Calculate the second derivative
50     ear_x_ddot = derivative(ear_x_dot ,time[0:len(time)-1])
51     ear_y_ddot = derivative(ear_y_dot ,time[0:len(time)-1])
52     halt_x_ddot = derivative(halt_x_dot ,time[0:len(time)-1])
53     halt_y_ddot = derivative(halt_y_dot ,time[0:len(time)-1])
54
55     # FFT
56
57     # X-axis for FFT plot
58     # Fs = 150.0;           # Sampling frequency
59     # Ts = 1.0/Fs;         # Sampling period

```

```

59 # t = np.arange(0,1,Ts) # time vector
60 #
61 # n = len(ear_x_filt)
62 # k = np.arange(n)
63 # T = n/Fs
64 # frq = k/T # two sides frequency range
65 # frq = frq[range(n/2.0)] # one side frequency range
66
67 # ear_x_fft = fft(ear_x_filt)/n
68 # Y = ear_x_fft[range(n/2)]
69
70 ear_x_fft = np.fft.fft(ear_x_filt)
71 ear_y_fft = np.fft.fft(ear_y_filt)
72 halt_x_fft = np.fft.fft(halt_x_filt)
73 halt_y_fft = np.fft.fft(halt_y_filt)
74
75 ear_x_dot_fft = np.fft.fft(ear_x_dot)
76 ear_y_dot_fft = np.fft.fft(ear_y_dot)
77 halt_x_dot_fft = np.fft.fft(halt_x_dot)
78 halt_y_dot_fft = np.fft.fft(halt_y_dot)
79
80 ear_x_ddot_fft = np.fft.fft(ear_x_ddot)
81 ear_y_ddot_fft = np.fft.fft(ear_y_ddot)
82 halt_x_ddot_fft = np.fft.fft(halt_x_ddot)
83 halt_y_ddot_fft = np.fft.fft(halt_y_ddot)
84
85 # Scatter plot.
86 fig = plt.figure()
87 plt.title("Ear Tip and Halter Position Over Time")
88 # plt.axis([0, frame_size[0], 0, frame_size[1]])
89 plt.xlabel("X Position (pixels)")
90 plt.ylabel("Y Position (pixels)")
91 plt.scatter(ear_x_filt, ear_y_filt, 2, c=time, cmap=cm.viridis)
92 cb = plt.colorbar(extend='max')
93 cb.set_label('Ear motion over time in milliseconds')
94 plt.scatter(halt_x_filt, halt_y_filt, 2, c=time, cmap=cm.plasma)
95 plt.gca().invert_yaxis()
96 cb = plt.colorbar(extend='max')
97 cb.set_label('Head motion over time in milliseconds')
98 fig.set_size_inches(14,6)
99 fig.savefig('../fft/' + csv_file + '_scatter_filt.png', dpi=150,
100             bbox_inches='tight')
101 # Ear plot.
102 #####

```

```

102 fig, axes = plt.subplots(3,1)
103 fig.suptitle("Position, Velocity, and Acceleration of Ear Point")
104 axes[0].plot(time, ear_x_filt, label='x', color="#660000")
105 axes[0].plot(time, ear_y_filt, label='y', color="#FF6600")
106 # Highlight point on graph
107 # rect = patches.Rectangle((10000,0), width=1000, height=2000,
108 #     color='Blue', alpha=0.5)
109 axes[0].set_ylabel('Position \n (pixels)')
110 axes[0].invert_yaxis()
111 axes[0].legend(loc='lower right')
112 ax0_1 = axes[0].twinx()
113 ax0_1.set_yticks([-1,1])
114 ax0_1.set_yticklabels(['Bottom, Right', 'Top, Left'])
115 # Plot the first derivative, velocity.
116 axes[1].plot(time[0:len(time)-1], ear_x_dot, label=r'$\dot{x}$',
117             color="#660000")
118 axes[1].plot(time[0:len(time)-1], ear_y_dot, label=r'$\dot{y}$',
119             color="#FF6600")
120 axes[1].set_ylabel('Velocity \n (pixels/ms)')
121 axes[1].set_ylim(-0.8, 0.8)
122 axes[1].invert_yaxis()
123 axes[1].legend(loc='lower right')
124 ax1_1 = axes[1].twinx()
125 ax1_1.set_yticks([-1,1])
126 ax1_1.set_yticklabels(['Bottom, Right', 'Top, Left'])
127 # Plot the second derivative, acceleration.
128 axes[2].plot(time[0:len(time)-2], ear_x_ddot, label=r'$\ddot{x}$',
129             color="#660000")
130 axes[2].plot(time[0:len(time)-2], ear_y_ddot, label=r'$\ddot{y}$',
131             color="#FF6600")
132 axes[2].set_ylabel('Acceleration \n (pixels/$ms^2$)')
133 axes[2].set_ylim(-0.01, 0.01)
134 axes[2].invert_yaxis()
135 axes[2].legend(loc='lower right')
136 axes[2].set_xlabel('time (ms)')
137 ax2_1 = axes[2].twinx()
138 ax2_1.set_yticks([-1,1])
139 ax2_1.set_yticklabels(['Bottom, Right', 'Top, Left'])
140 fig.set_size_inches(14,6)
141 fig.savefig('../fft/' + csv_file + '_ear_filt.png', dpi=150,
142             bbox_inches='tight')
143
144 ## Halter point.
145 #####
146 fig, axes = plt.subplots(3,1)

```

```

141 fig.suptitle("Position, Velocity, and Acceleration of Halter Point"
142 )
143 axes[0].plot(time, halt_x_filt, label='x', color="#660000")
144 axes[0].plot(time, halt_y_filt, label='y', color="#FF6600")
145 # axes[0].annotate('Tracks different halter point', xy=(2000, 900),
146 xytext=(6000, 400),
147 # arrowprops=dict(arrowstyle="->", connectionstyle="angle,
148 angleA=0, angleB=90, rad=5"))
149 # axes[0].annotate('Frame 121', xy=(4070, 730), xytext=(7000, 1100)
150 # arrowprops=dict(arrowstyle="->", connectionstyle="angle,
151 angleA=0, angleB=-90, rad=5"))
152 # Highlight point on graph
153 # rect = patches.Rectangle((200,0), width=3300, height=2000,
154 # color='Blue', alpha=0.5)
155 # axes[0].add_patch(rect)
156 # rect2 = patches.Rectangle((4700,-10), width=1500, height=2000,
157 # color='green', alpha=0.5)
158 # axes[0].add_patch(rect2)
159 # rect3 = patches.Rectangle((13000,-10), width=7000, height=2000,
160 # color='green', alpha=0.5)
161 # axes[0].add_patch(rect3)
162 axes[0].set_ylabel('Position \n (pixels)')
163 axes[0].invert_yaxis()
164 axes[0].legend(loc='lower right')
165 ax0_1 = axes[0].twinx()
166 ax0_1.set_yticks([-1,1])
167 ax0_1.set_yticklabels(['Bottom, Right', 'Top, Left'])
168 # Plot the first derivative, velocity.
169 axes[1].plot(time[0:len(time)-1], halt_x_dot, label=r'$\dot{x}$',
170 color="#660000")
171 axes[1].plot(time[0:len(time)-1], halt_y_dot, label=r'$\dot{y}$',
172 color="#FF6600")
173 # Highlight point on graph
174 # rect = patches.Rectangle((200,-10), width=3300, height=2000,
175 # color='Blue', alpha=0.5)
176 # axes[1].add_patch(rect)
177 # rect2 = patches.Rectangle((4700,-10), width=1500, height=2000,
178 # color='green', alpha=0.5)
179 # axes[1].add_patch(rect2)
180 # rect3 = patches.Rectangle((13000,-10), width=7000, height=2000,
181 # color='green', alpha=0.5)
182 # axes[1].add_patch(rect3)
183 axes[1].set_ylabel('Velocity \n (pixels/ms)')
184 axes[1].set_ylim(-0.8, 0.8)

```

```

179 axes[1].invert_yaxis()
180 axes[1].legend(loc='lower right')
181 ax1_1 = axes[1].twinx()
182 ax1_1.set_yticks([-1,1])
183 ax1_1.set_yticklabels(['Bottom, Right', 'Top, Left'])
184 # Plot the second derivative, acceleration.
185 axes[2].plot(time[0:len(time)-2], halt_x_ddot, label=r'$\ddot{x}$',
186             color="#660000")
187 axes[2].plot(time[0:len(time)-2], halt_y_ddot, label=r'$\ddot{y}$',
188             color="#FF6600")
189 # Highlight point on graph
190 # rect = patches.Rectangle((200,-10), width=3300, height=2000,
191 # color='Blue', alpha=0.5)
192 # axes[2].add_patch(rect)
193 # rect2 = patches.Rectangle((4700,-10), width=1500, height=2000,
194 # color='green', alpha=0.5)
195 # axes[2].add_patch(rect2)
196 # rect3 = patches.Rectangle((13000,-10), width=7000, height=2000,
197 # color='green', alpha=0.5)
198 # axes[2].add_patch(rect3)
199 axes[2].set_ylabel('Acceleration \n (pixels/$ms^2$)')
200 axes[2].set_ylim(-0.01, 0.01)
201 axes[2].invert_yaxis()
202 axes[2].legend(loc='lower right')
203 axes[2].set_xlabel('time (ms)')
204 ax2_1 = axes[2].twinx()
205 ax2_1.set_yticks([-1,1])
206 ax2_1.set_yticklabels(['Bottom, Right', 'Top, Left'])
207 fig.set_size_inches(14,6)
208 fig.savefig('../fft/' + csv_file + '_halter_filt.png', dpi=150,
209             bbox_inches='tight')
210
211 # FFT
212 #####
213
214 # Set up x axis for FFT
215
216 Fs = 30; # sampling rate
217 # Ts = 1.0/Fs; # sampling interval
218 # t = np.arange(0,1,Ts) # time vector
219
220 n = len(ear_x_filt) # length of the signal
221 k = np.arange(n)
222 T = n/Fs
223 frq = k/T # two sides frequency range

```

```

219     frq = frq[range(int(n/2))] # one side frequency range
220
221     ear_x_fft = ear_x_fft[range(int(n/2))]
222     ear_y_fft = ear_y_fft[range(int(n/2))]
223     halt_x_fft = halt_x_fft[range(int(n/2))]
224     halt_y_fft = halt_x_fft[range(int(n/2))]
225
226     ear_x_dot_fft = ear_x_dot_fft[range(int(n/2))]
227     ear_y_dot_fft = ear_y_dot_fft[range(int(n/2))]
228     halt_x_dot_fft = halt_x_dot_fft[range(int(n/2))]
229     halt_y_dot_fft = halt_y_dot_fft[range(int(n/2))]
230
231     ear_x_ddot_fft = ear_x_ddot_fft[range(int(n/2))]
232     ear_y_ddot_fft = ear_y_ddot_fft[range(int(n/2))]
233     halt_x_ddot_fft = halt_x_ddot_fft[range(int(n/2))]
234     halt_y_ddot_fft = halt_y_ddot_fft[range(int(n/2))]
235
236     # freq = np.fft.fftfreq(ear_x_fft.shape[-1])
237
238     # Ear FFT plot
239     #####
240     fig, axes = plt.subplots(3,1)
241     fig.suptitle("FFT of Position, Velocity, and Acceleration for the
242     Ear")
243     # axes[0].plot(freq, abs(y), label='y', color="#FF6600")
244     axes[0].plot(freq, abs(ear_x_fft), label='x', color="#660000")
245     axes[0].plot(freq, abs(ear_y_fft), label='y', color="#FF6600")
246     # Highlight point on graph
247     # rect = patches.Rectangle((10000,0), width=1000, height=2000,
248     #     color='Blue', alpha=0.5)
249     axes[0].set_ylabel('Position FFT \n Magnitude')
250     # axes[0].invert_yaxis()
251     axes[0].legend(loc='lower right')
252     # ax0_1 = axes[0].twinx()
253     # ax0_1.set_yticks([-1,1])
254     # ax0_1.set_yticklabels(['Bottom, Right', 'Top, Left'])
255     # Plot the first derivative, velocity.
256     axes[1].plot(freq, abs(ear_x_dot_fft), label=r'$\dot{x}$', color="
257     #660000")
258     axes[1].plot(freq, abs(ear_y_dot_fft), label=r'$\dot{y}$', color="#
259     FF6600")
260     axes[1].set_ylabel('Velocity FFT \n Magnitude')
261     # axes[1].set_ylim(-0.8, 0.8)
262     # axes[1].invert_yaxis()
263     axes[1].legend(loc='lower right')

```



```

260 # ax1_1 = axes[1].twinx()
261 # ax1_1.set_yticks([-1,1])
262 # ax1_1.set_yticklabels(['Bottom, Right', 'Top, Left'])
263 # Plot the second derivative, acceleration.
264 axes[2].plot(freq, abs(ear_x_ddot_fft), label=r'$\ddot{x}$', color="
    #660000")
265 axes[2].plot(freq, abs(ear_y_ddot_fft), label=r'$\ddot{y}$', color="
    #FF6600")
266 axes[2].set_ylabel('Acceleration FFT \n Magnitude')
267 # axes[2].set_ylim(-0.03, 0.03)
268 # axes[2].invert_yaxis()
269 axes[2].legend(loc='lower right')
270 axes[2].set_xlabel('Freq (Hz)')
271 fig.set_size_inches(14,6)
272 fig.savefig('../fft/' + csv_file + '_ear_fft.png', dpi=150,
    bbox_inches='tight')
273
274 ## Halter FFT plot
    #####
275 fig, axes = plt.subplots(3,1)
276 fig.suptitle("FFT of Position, Velocity, and Acceleration for the
    Halter")
277 axes[0].plot(freq, abs(halt_x_fft), label='x', color="#660000")
278 axes[0].plot(freq, abs(halt_y_fft), label='y', color="#FF6600")
279 # axes[0].annotate('Tracks different halter point', xy=(2000, 900),
    xytext=(6000, 400),
280 #     arrowprops=dict(arrowstyle="->", connectionstyle="angle,
    angleA=0, angleB=90, rad=5"))
281 # axes[0].annotate('Frame 121', xy=(4070, 730), xytext=(7000, 1100)
    ,
282 #     arrowprops=dict(arrowstyle="->", connectionstyle="angle,
    angleA=0, angleB=-90, rad=5"))
283 # Highlight point on graph
284 # rect = patches.Rectangle((200,0), width=3300, height=2000,
285 #     color='Blue', alpha=0.5)
286 # axes[0].add_patch(rect)
287 # rect2 = patches.Rectangle((4700,-10), width=1500, height=2000,
288 #     color='green', alpha=0.5)
289 # axes[0].add_patch(rect2)
290 # rect3 = patches.Rectangle((13000,-10), width=7000, height=2000,
291 #     color='green', alpha=0.5)
292 # axes[0].add_patch(rect3)
293 axes[0].set_ylabel('Position FFT \n Magnitude')
294 axes[0].legend(loc='lower right')
295 # Plot the first derivative, velocity.

```

```

296 axes[1].plot(freq, abs(halt_x_dot_fft), label=r'\dot{x}$', color="
      #660000")
297 axes[1].plot(freq, abs(halt_y_dot_fft), label=r'\dot{y}$', color="#
      FF6600")
298 # Highlight point on graph
299 # rect = patches.Rectangle((200,-10), width=3300, height=2000,
300 #     color='Blue', alpha=0.5)
301 # axes[1].add_patch(rect)
302 # rect2 = patches.Rectangle((4700,-10), width=1500, height=2000,
303 #     color='green', alpha=0.5)
304 # axes[1].add_patch(rect2)
305 # rect3 = patches.Rectangle((13000,-10), width=7000, height=2000,
306 #     color='green', alpha=0.5)
307 # axes[1].add_patch(rect3)
308 axes[1].set_ylabel('Velocity FFT \n Magnitude')
309 # axes[1].set_ylim(-0.8, 0.8)
310 axes[1].legend(loc='lower right')
311 # Plot the second derivative, acceleration.
312 axes[2].plot(freq, abs(halt_x_ddot_fft), label=r'\ddot{x}$', color=
      "#660000")
313 axes[2].plot(freq, abs(halt_y_ddot_fft), label=r'\ddot{y}$', color=
      "#FF6600")
314 # Highlight point on graph
315 # rect = patches.Rectangle((200,-10), width=3300, height=2000,
316 #     color='Blue', alpha=0.5)
317 # axes[2].add_patch(rect)
318 # rect2 = patches.Rectangle((4700,-10), width=1500, height=2000,
319 #     color='green', alpha=0.5)
320 # axes[2].add_patch(rect2)
321 # rect3 = patches.Rectangle((13000,-10), width=7000, height=2000,
322 #     color='green', alpha=0.5)
323 # axes[2].add_patch(rect3)
324 axes[2].set_ylabel('Acceleration FFT \n Magnitude')
325 axes[2].legend(loc='lower right')
326 axes[2].set_xlabel('Freq (Hz)')
327 fig.set_size_inches(14,6)
328 fig.savefig('./fft/' + csv_file + '_halter_fft.png', dpi=150,
      bbox_inches='tight')
329
330 with open(folder_location + csv_file + extension) as csvfile:
331     readCSV = csv.DictReader(csvfile, delimiter=',')
332
333     time = []
334     frame_num = []
335     ear_x = []

```

```
336     ear_y = []
337     halt_x = []
338     halt_y = []
339     ear_x_zero = []
340     ear_y_zero = []
341
342     for row in readCSV:
343         time.append(float(row['Time']))
344         frame_num.append(float(row['Frame Number']))
345         ear_x.append(float(row['Ear X']))
346         ear_y.append(float(row['Ear Y']))
347         halt_x.append(float(row['Halter X']))
348         halt_y.append(float(row['Halter Y']))
349
350 # graph_data(time, ear_x, ear_y, halt_x, halt_y, size)
351
352 # set position to zero start
353 ear_x_zero = ([i-ear_x[0] for i in ear_x])
354 ear_y_zero = ([i-ear_y[0] for i in ear_y])
355 halt_x_zero = ([i-halt_x[0] for i in halt_x])
356 halt_y_zero = ([i-halt_y[0] for i in halt_y])
357
358 # ear_x_zero = ear_x - ear_x[0]
359 # print(ear_x)
360 # print(ear_x_zero)
361 # ear_y_zero = [i-ear_y[0] for i in ear_y]
362 # halt_x_zero = [i-halt_x[0] for i in halt_x]
363 # halt_y_zero = [i-halt_x[0] for i in halt_x]
364
365 graph_data(time, ear_x_zero, ear_y_zero, halt_x_zero, halt_y_zero, size
366 )
367 plt.show()
```