Cervical Proprioception Motion Analyzer

Product Readiness Report

Department of Engineering and Physics
ENGR 4201. Senior Design
Prof. Wells, Dr. Olree, Dr. Gibson.

Ethan Grimes, Nathan Vielmette, George Cook and Keneth Chelelgo

April 27th, 2021

Table of Contents

As Tested Design

Introduction

Every year in the United States, about 10 million people go to Physical Therapy. Of these, about 20% experience some kind of cervicogenic or neck related condition. This could be whiplash, stroke, traumatic brain injury, or even Parkinson's disease. When they come in to be treated, they're subjected to a battery of different tests. 5/14 of these don't use any kind of numerics standards, and the remainder use very poor metrology. The gold standard of these tests is what's known as the Cervical Positioning Error test, or CPE test. The way this test is conducted now is still very crude, and doesn't provide much objective information or data. Physical Therapists are essentially making diagnoses based on their best intuition. The CervPro team decided that a device that aids the Physical Therapist in collecting objective data for the CPE test would be a good project to pursue.

Customer Needs

After conducting several interviews with multiple physical therapists, needs that were expressed included the following: The device needs to be easy to use. Physical therapists may not be as technologically savvy as the engineers who create the device, so it needs to output only the necessary information. Dr. Galindo expressed, "Typically, PTs will choose tests and measures based on convenience (i.e. if the 'best' evaluation tool takes me 30 minutes but a 'good' evaluation tool that is valid/reliable takes me only 10min, I will always chose the 10min version.)" They expressed that however the device operates, comfort is very important for the patient. Additionally, the device should not inhibit the patient's motions. The device should provide objective, clear, and accurate data. Finally, the device needs to comply with all relevant legal and industry standards in order to be usable in a medical setting.

Technical Specifications

The time required to run a CPE test with the CervPro device will not exceed 10 minutes. This is double the time required to run the CPE test without the device. The proportion of therapists able to make a diagnosis exclusively using the device and within 40 seconds of completing the CPE test using the device shall be above 50%. Any device hardware will not inhibit the movement of the neck across an 80 degree RoM about all rotational axes. Also, the weight of any device attached to the patient will be less than 10 lbs.

The CervPro device shall measure the angular position of the head across an ideal 80° and acceptable 60° RoM in either direction with error of less than 5%. The device will measure angular velocity of the head up to an ideal 60° per second and an acceptable 50° per second with an error of less than 5%. The position of any hardware of the device (if located on the patient's head) will not shift more than 5° during 60° per second rotational speed. Any data collected will be saved to file while immediately displaying a therapist-defined output.

The CervPro device will not require Personal Protective Equipment (PPE) for operation.It will not emit sound louder than 60 dB, and any data transmission shall comply with relevant FCC and medical device regulations.

Overall System Design

The CervPro device is a completely electronic device with no moving parts. The overall goal of the device is to measure patients' head movements in every degree of freedom. The data capturing would be done by a single camera that records the patient performing the CPE test. The data is then sent to the microprocessor via USB cable to be analyzed. The microprocessor will run a program that uses facial recognition techniques to understand where the patients' heads are in space and then will output movement information to the physical therapists. The device will be powered by plugging into a standard wall outlet.

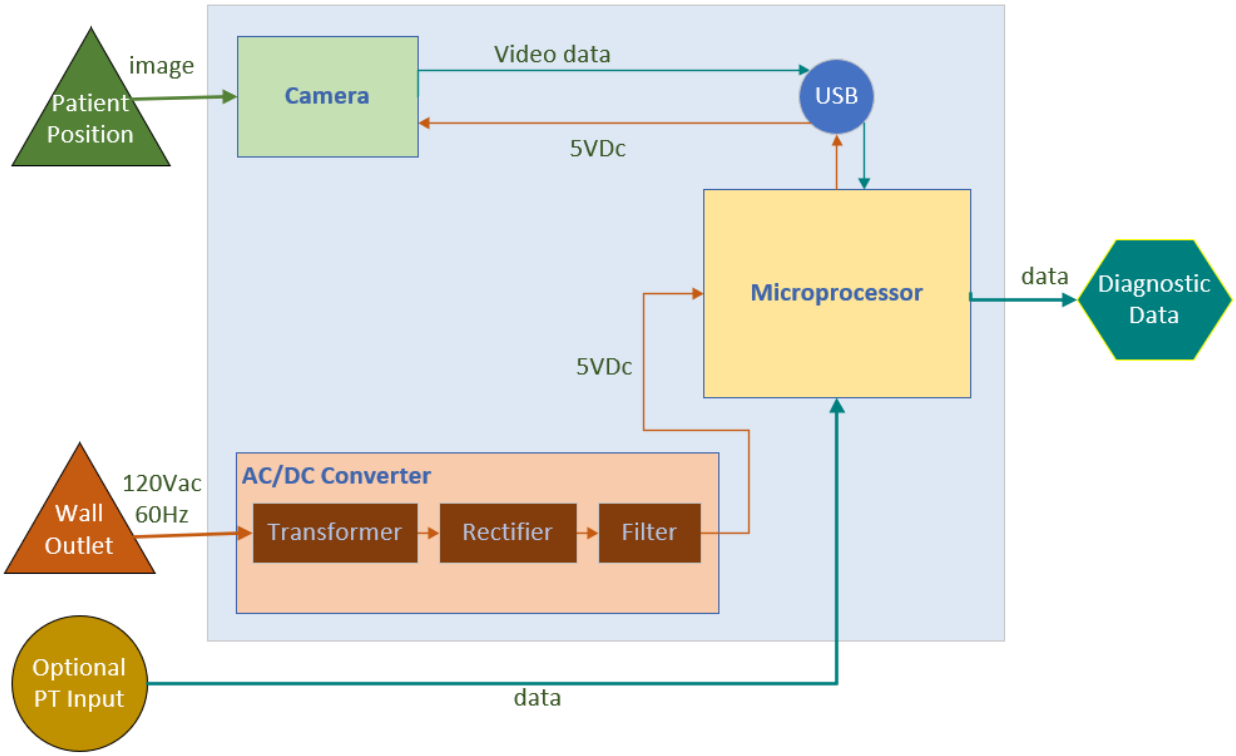


Figure 1. Functional Decomposition Diagram of the CervPro Device

Shown above is a functional decomposition diagram of the CervPro device. The overall system is fairly simple compared to some other senior design projects. There are 3 main subsystems in the device. Two of these subsystems require actual engineering. The camera subsystem has been purchased, not designed. However, the microprocessor and power system have been designed and will be fabricated next semester.

There will be 3 inputs that will be going into the device, the patients' head positions, power from the wall outlet the device will be plugged into, and some optional input by the physical therapist. This optional input hasn't been defined yet, but it will most likely be some sort of preference specification that the physical therapist will express to analyze the patients' head movements in a more specific way.

As shown below in figure 2, there are two human faces that the facial recognition program is finding. The program is tracing specific facial features with lime green lines to indicate recognition. This is an example of the program working on a single picture, but the program would be able to run this kind of analysis on video footage coming in at 30 frames per second. The program will then run some calculations based on the distances between certain pixels to find the closest approximation of where the person's head is in space numerically. These calculations will be based on the position of the patient's head at the beginning of their CPE test.

The power system in the CervPro device as mentioned before will draw power from a standard wall outlet. What's important to note conceptually about this diagram is that the power supply has three main components: a step down transformer, a rectifier, and a filter. The power supply will provide 5V of power to both the microprocessor and the camera.

Figure 2. Example of facial recognition program working on human faces

The physical device is an interaction of several different components. The housing was ordered, and specially picked to properly fit all the components while supplying input and output holes and a see through top. The housing chosen was the SK-19 enclosure. The camera chosen was the Playstation Eye, which was selected for its price point, and frame rate and resolution compatibility with the microprocessor. The camera plugs in externally so that the camera can be moved wherever the physical therapist needs it. A small LCD touchscreen was also picked to operate as the component that the physical therapist interacts with. The physical therapist will start and stop the program using the screen and will see data on it as well.

Figure 3. SK-19 Enclosure

Figure 4. Playstation Eye camera

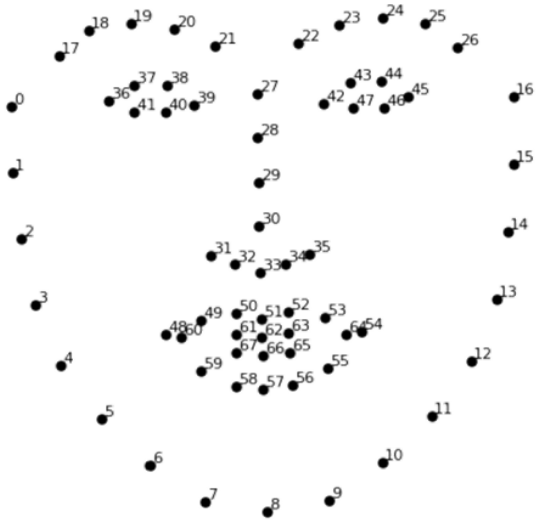
Figure 5. LCD Touch Screen

Computer Engineering Design

The computer subsystem was the most complex portion of the entire system. The system had certain hardware and software requirements to be able to run facial recognition: an operating system, fast cpu, multiple usb ports, and an internet connection.

The internet connection was needed to install the needed libraries and CMake to compile them. These libraries were OpenCV and dlib. dlib is a standard C++ library that could have easily run on other, less powerful, microprocessors but OpenCV required an operating system to perform some of the needed communications with the camera. The operating system helped in creating an easy and effective user interface for physical therapists to communicate with the program; this user interface was implemented by using the GUI libraries found in dlib. dlib also provided the core of the program by performing the facial recognition and returning multiple points along the face, seen in figure C1, where point 33 on the nose is used for calculations: the points for each frame is saved (done on a separate thread); every 15 frames the angular velocity is calculated; this repeats till the end of the test; when the stop button is pressed, the program waits for all of the threads to finish processing and then will find the furthest left and right points and calculate the overall angular velocity of the patient's head; finally it compares the first and last frames to see how far of the individual is from the starting point (ass seen in the flow chart in Appendix B and state chart in Appendix C).

The process of running the facial recognition requires a lot of power to perform quickly and efficiently. Some existing tests on the speed needed to perform the testing had been performed (as seen in Appendix D) and was used in choosing a microprocessor. As we tested the system we learned that the speed requirement was not a linear growth. In use the frames per second processed by the microprocessor was below 60 fps which is what we were expecting from the existing tests if the requirements were linear growth. We also looked at different kinds of facial recognition and found four major types: Haar, DNN, HoG, and MMOD. These four options exist within OpenCV and dlib. Through some more existing testing we saw that the HoG facial recognition algorithm was the fastest CPU based technique, and that the only faster technique was MMOD with a GPU. With this we implemented our project with the HoG algorithm.



- Jaw Points = 0–16
- Right Brow Points = 17–21
- Left Brow Points = 22–26
- Nose Points = 27–35
- Right Eye Points = 36–41
- Left Eye Points = 42–47
- Mouth Points = 48–60
- Lips Points = 61–67

Figure 6. Points found by dlib HoG facial recognition [1].

Electrical Engineering Design

The overall design was created in three portions: a transformer, rectifier, and filter. The individual portions and the overall functions were simulated in multisim. Similarly, the three pieces were tested individually to meet their needed functionality.

The transformer needed a voltage ratio within 1:20 to 1:4 at the frequency of 60Hz (in order to provide between 6-25V to the regulator), this was verified experimentally as a gain of 7.25, which approximates the stated one of 7.5 and is within the needed tolerance. In order to show that the device would maintain integrity in non-ideal conditions (although NEMA standards are very strict), including a stepped down 50Hz source (e.g. some Japanese mains),

the transformer gain characteristics were run from 6Hz up to 600Hz (one decade above and below intended) to ensure no significant frequency gain difference exists as shown in Figure 7.

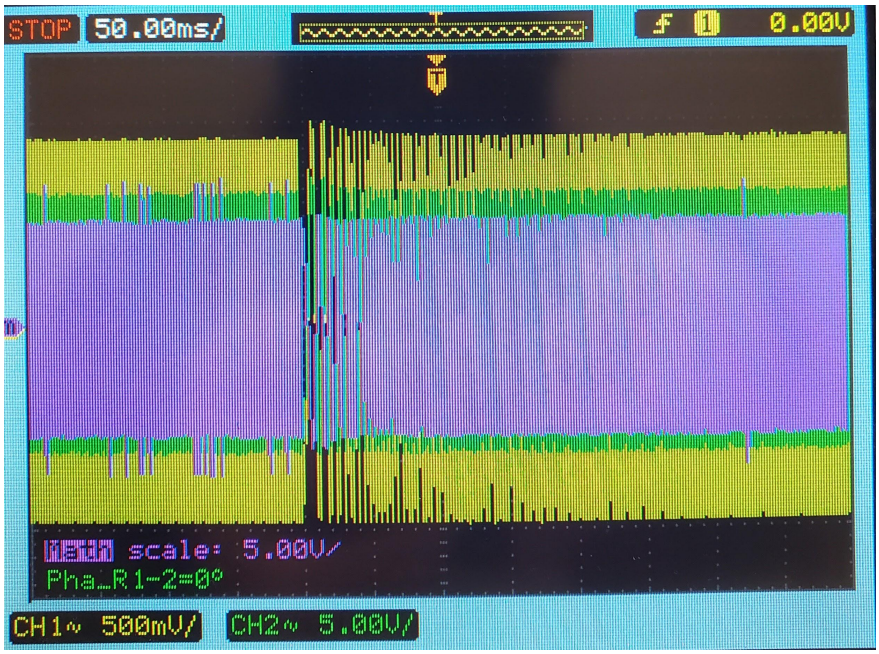


Figure 7. Sweep characteristics from 6 to 600Hz

The rectifier must rectify a 60Hz 25Vrms input with ideally less than 1% cycle lag and negative voltage. The first stat checking lag was ascertained by offsetting an input voltage cycle so that the entire input voltage was positive and superimposing it onto the output voltage (i.e. adding a slight offset for the voltage drop ~0.5V) as shown in Figure 8.

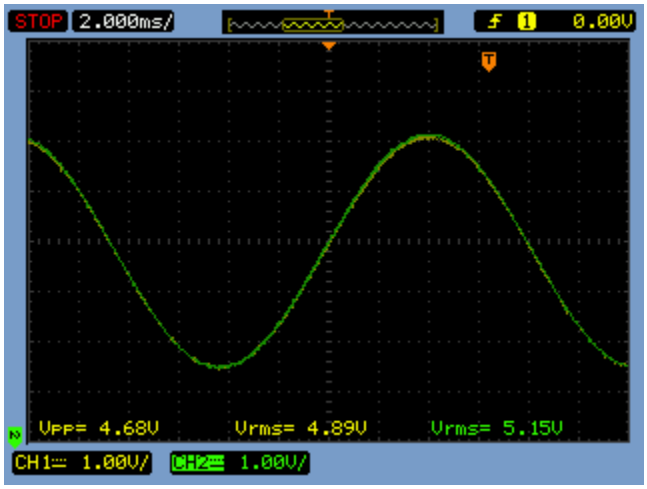


Figure 8. Rectifier lag check

After this, the rectifier was subject to 60Hz 7Vrms (no offset) input, and the output voltage varied from a maximum 9.41V to a minimum of -160mV (not quite within 1%, but still acceptable in practice).

The filter itself had to output a constant 5±.1Vdc with less than 5% ripple (250mV) and controlled current from a rectified input voltage. Because the final regulator dissipates excess supplied power as heat, it is important to control both the peak and RMS voltage to it, so the initial stage utilizes a large capacitor and negative feedback circuit using much higher rated power transistors to bring the peak and RMS voltages as near to each other and 5V as possible as shown in Figure 10. Due to this design, it cannot be tested easily without the rectifier; even with this in place supplying less voltage than mains, it can output the rated 5V as shown in Figure 9.

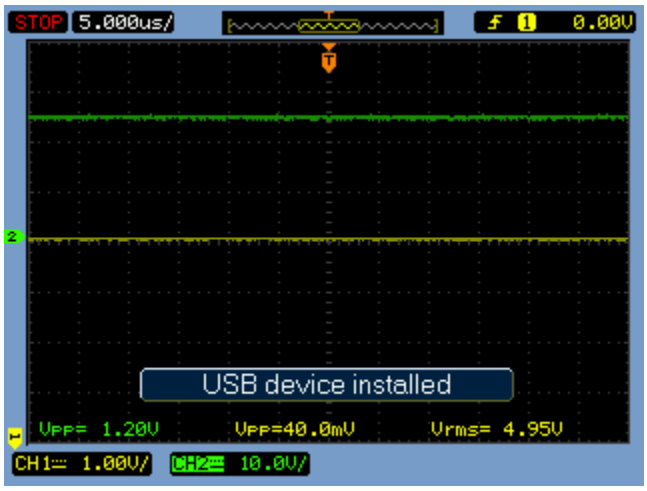


Figure 9. Stage 1 (Ch2) and Stage 2(Ch1) outputs of filter

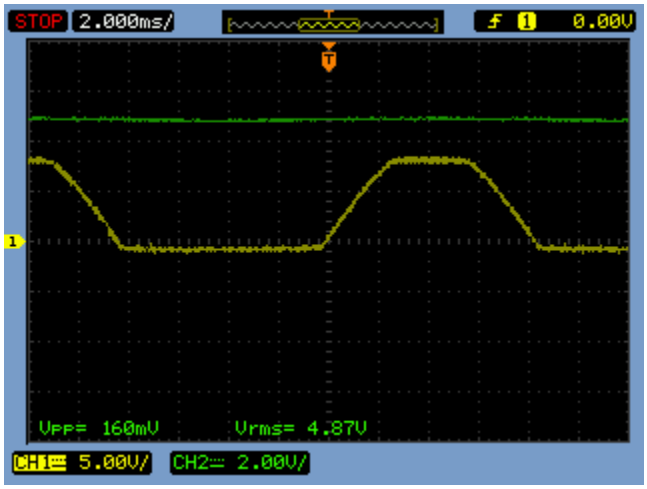


Figure 10. Underpowered (Ch2) filter results showing near 5V output (Ch1)

This configuration maintains a ripple that is small due to the regulator's designed ability to dissipate additional energy; because of this, a heat sink is attached to it. The entire configuration was plugged into a NEMA 5 outlet with an inline GFCI and functioned as simulated and demonstrated piecemeal as shown in Figures 11. and 12.

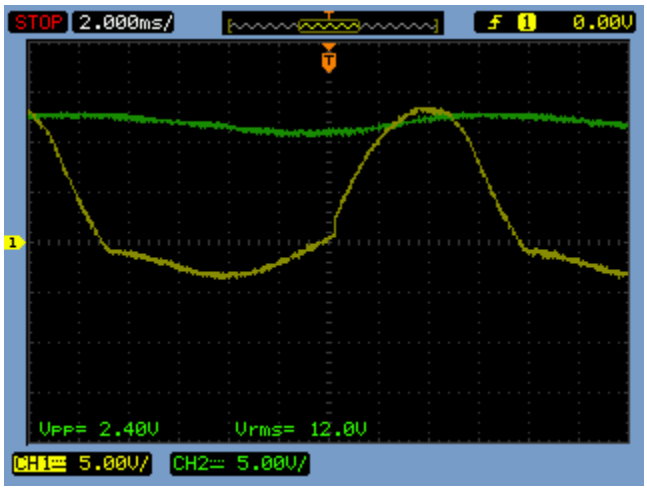


Figure 11. Measurements after rectification (Ch 1) and stage 1 of filter (Ch 2)

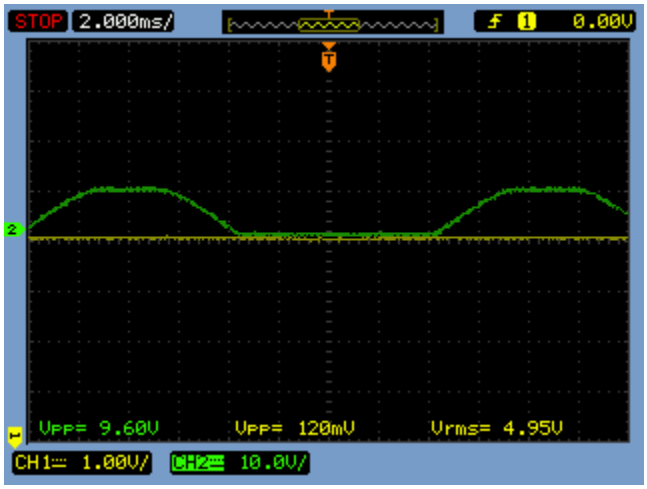


Figure 12. Measurements after rectification (Ch 1) and system output (Ch 2)

The PCB design itself was driven by several needs known from the start: robust conductivity to reduce output impedance losses, minimal footprint to fit within the package, smoothest trace routings to prevent EM radiation creation, and maximized spacing between conductive surfaces to prevent possible hazardous discharges. After failures present in previous versions, it was identified that trace and pad smoothness was of priority as sharp concavities can cause manufacturing defects with the available Protomat. Additionally, a Control of

Hazardous Energy (“CoHE”) failure occurred during a testing stage when a teammate unknowingly discharged the energy between two points he could not recall.

Control of Hazardous Energies is of primary importance, so while from the outset it was prioritized, the hazard presented by CoHE failures is potentially lethal to the user and certainly so for the circuit. The dielectric strength of air is 3kV/mm, so for a circuit with maximum possible potential difference of 16V, the spark gap would be 5.3µm. The smallest spacing was between the pads of the TO92 package transistor, which measures 59µm. After the Mk IIIa board had this catastrophic failure which destroyed the subsystem’s ability to function, failure analysis was performed on the components and it was discovered the regulator had shorted onto something else. The process of analyzing the components ultimately destroyed the board itself. Changes beyond that point incorporated potting with ethylene-vinyl acetate thermoplastic adhesive (which has a dielectric strength roughly one-half that of air), moving the TO-220 and 247 components such that their conductive faces are further away from other charged surfaces, and creating two isolated and clearly understandable input and output point pairs. Ever since the changes were implemented, testing involving non-electrical engineers using the subsystem has been wholly successful.

Figure 13. Mk IIIc PCB as printed

Mechanical Engineering Design

The program captures the location of the head through a multi-step process. Before starting, the user sees video feedback of the camera's perspective--looking at the patient's face which is positioned 90cm away from the camera. First, the user will click the start button on the touch screen of the device when they are satisfied with the patient's position as being the starting position. The program at this point records the pixel coordinates of the tip of the patient's nose and logs it for calculations. Seen below, the red dot shows the point that the program saves.

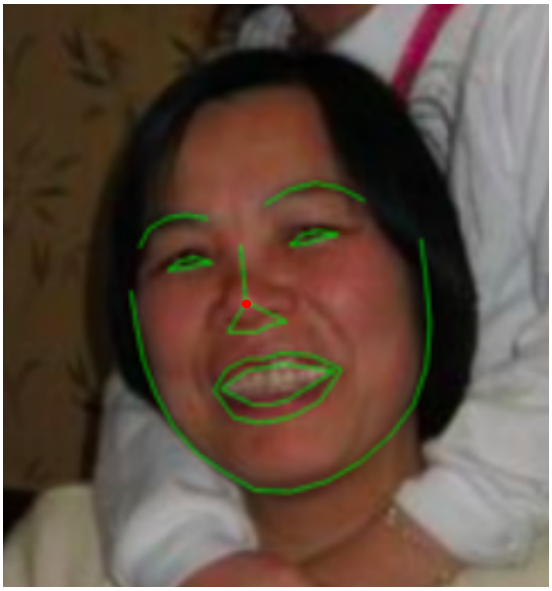


Figure 14. Point recorded by the program shown in red

After the initial point has been recorded, the physical therapist will guide the patient through the desired CPE movements. During this whole process, the program is recording the positions of the tip of the nose every frame. The differences in these positions are then divided by the time to yield angular velocity. The angular velocity is found by dividing the difference over the time of 15 frames or 0.25 seconds. Once the patient is done with their movements, and has trained their gaze on where they believe the center position to be with their eyes closed, the Physical Therapist will stop the program. At this point, the program records the final position of the patient at the tip of the nose.

Once the program has the starting and ending positions of the tip of the nose in pixel coordinates as seen by the screen, the euclidean distance between these two points is found.

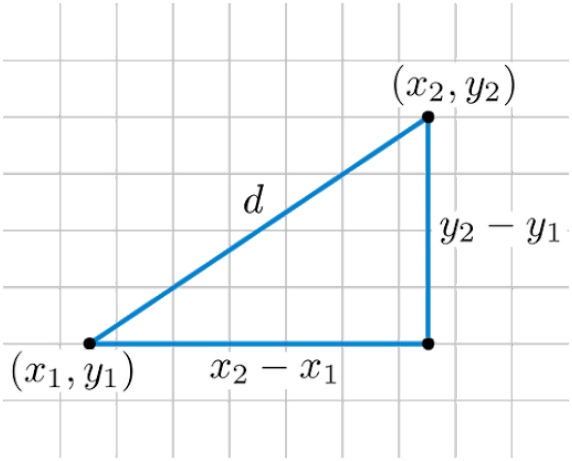


Figure 15. Graphic showing Euclidean distance

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

Once this Euclidean distance is found in pixel screen coordinates , it is then converted to physical centimeters as seen on the face by using the equation

$$d_{cm} = \frac{(90\ cm)(d_{pc})(0.635)}{(0.413774)(480)}$$

Where d_{cm} is the distance in centimeters, and d_{pc} is the distance in pixel count.

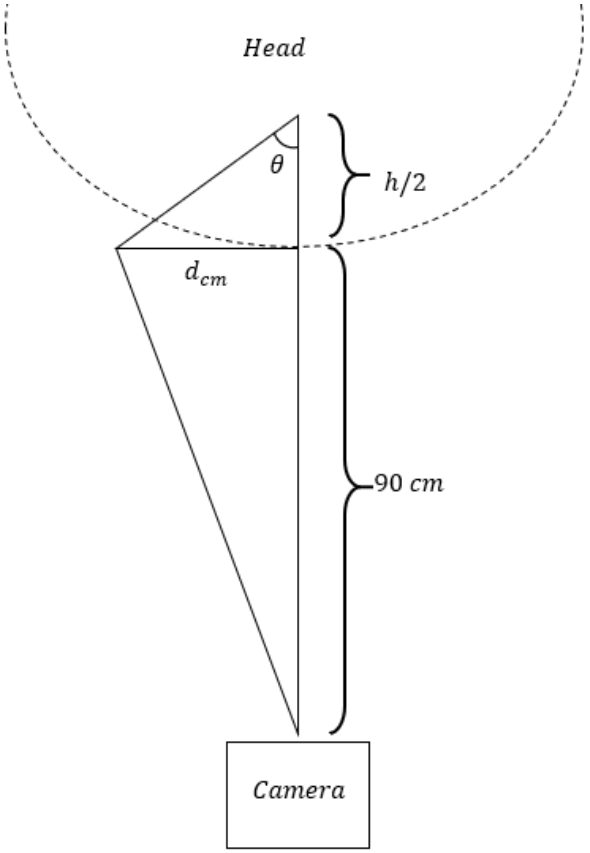


Figure 16. Figure showing geometry between head and camera.

$$\theta = tan^{-1}(d_{cm}/(h/2))$$

θrepresents the final angular difference between the patient's start and end position, and h/2 represents the radius of the patient's head, which can either be measured or assumed based on the Physical Therapist's desires for ease of use.

Through experimentation and testing which will be discussed later, it was discovered that the results that the program yielded were outside the desired 5% error range. In an effort to reduce error, experimental data was analyzed and a best-fit 7th order polynomial was created in MATLAB to show the relationship between the desired and actual angular displacement values. This polynomial was then used to calibrate the results. The x-axis is the measured values and the y-axis is the desired value. The trendline shows the relationship between the two.

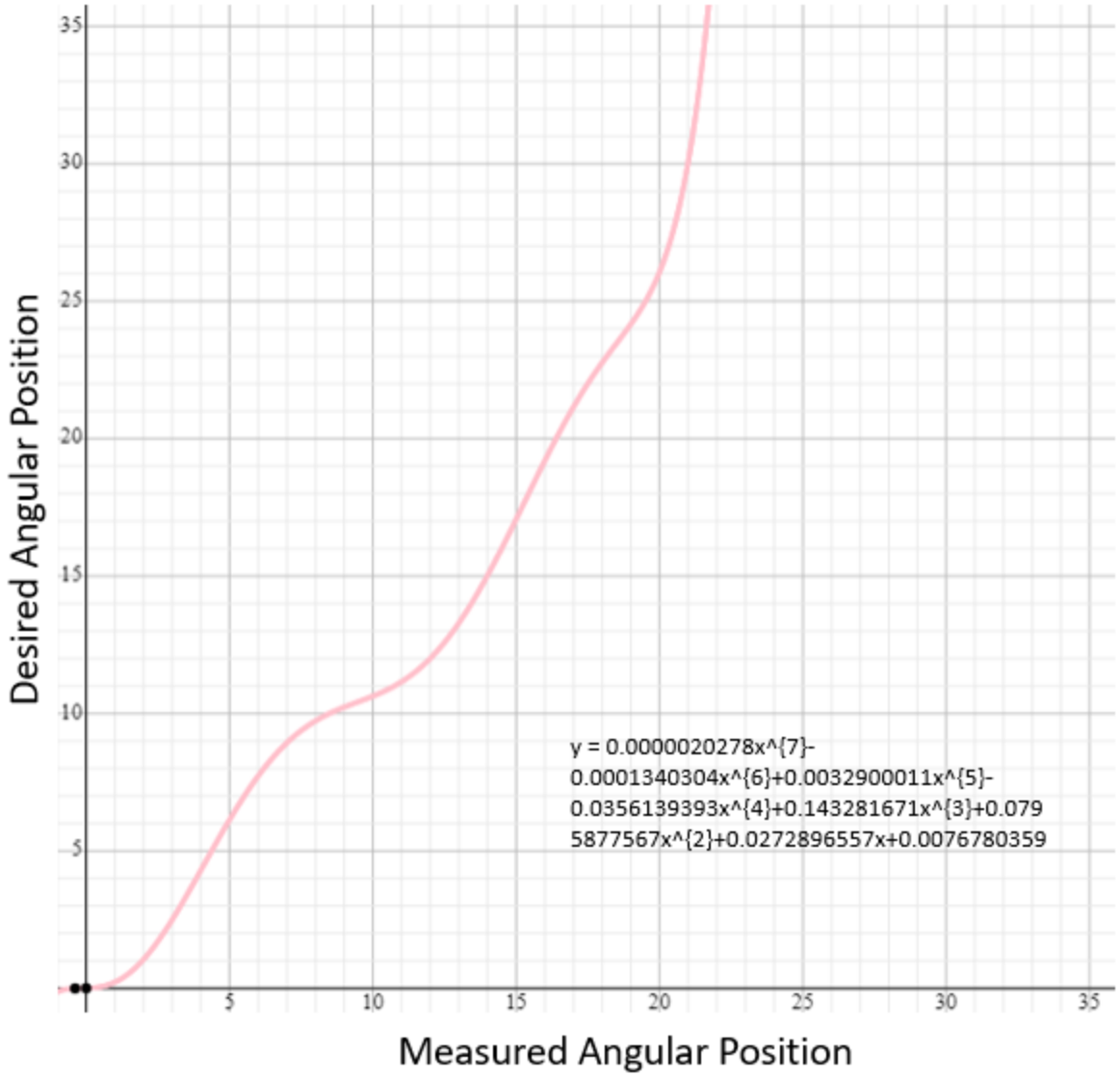


Figure 17. Best fit curve of relationship between measured and desired angular position

The cooling subsystem has been completed according to schedule. The microprocessor came with a stock heat sink with aluminum fins, and SolidWorks thermal studies showed it would not sufficiently dissipate the heat generated by the microprocessor. Other heat sink options were explored, and the ICE Cooling tower met the needed cooling requirements. These requirements were set in order for the microprocessor to operate at maximum efficiency.

The goal of the testing was to determine the effectiveness of the ICE cooling tower when compared to the stock heatsink. The temperatures were measured via IR thermometer, and the temperature was measured where the ICE cooling tower interfaces with the chip. These tests were conducted while loading the microprocessor with a higher level of computing power

than would be seen in a normal CPE test. The facial recognition algorithm was running as well as four tabs that were open streaming HD video on YouTube.

The testing began by turning off the Jeston desktop and disconnecting the power for 30 minutes. This allowed the microprocessor to cool and reach a steady state ambient temperature. The power was then connected and the system was booted up. For both procedures, the temperatures were measured at the same regular intervals. See the appendix for the results data. The ICE cooling tower can be seen installed in Figure 18.

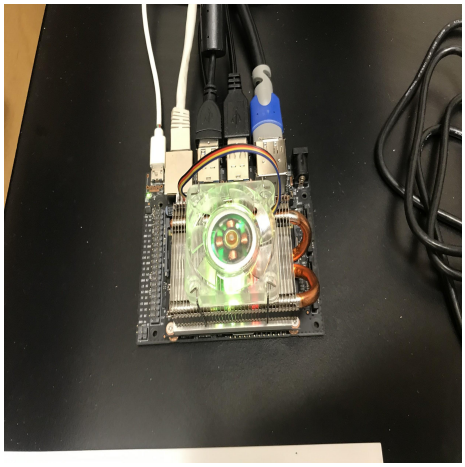


Figure 18. Nvidia Jetson Nano with the ICE cooling tower installed

The data collected was transformed into visual data analysis by plotting a graph of time vs temperature variations as shown in Figure 18. The testing results indicated that the temperature increased steadily on heatsinks to a maximum of 52.1 ℃ within 23 minutes of testing time.

For the ICE Cooling tower, the temperature increased while fluctuating, but at a slower rate to a maximum of 35.0℃ within 23 minutes of testing time.

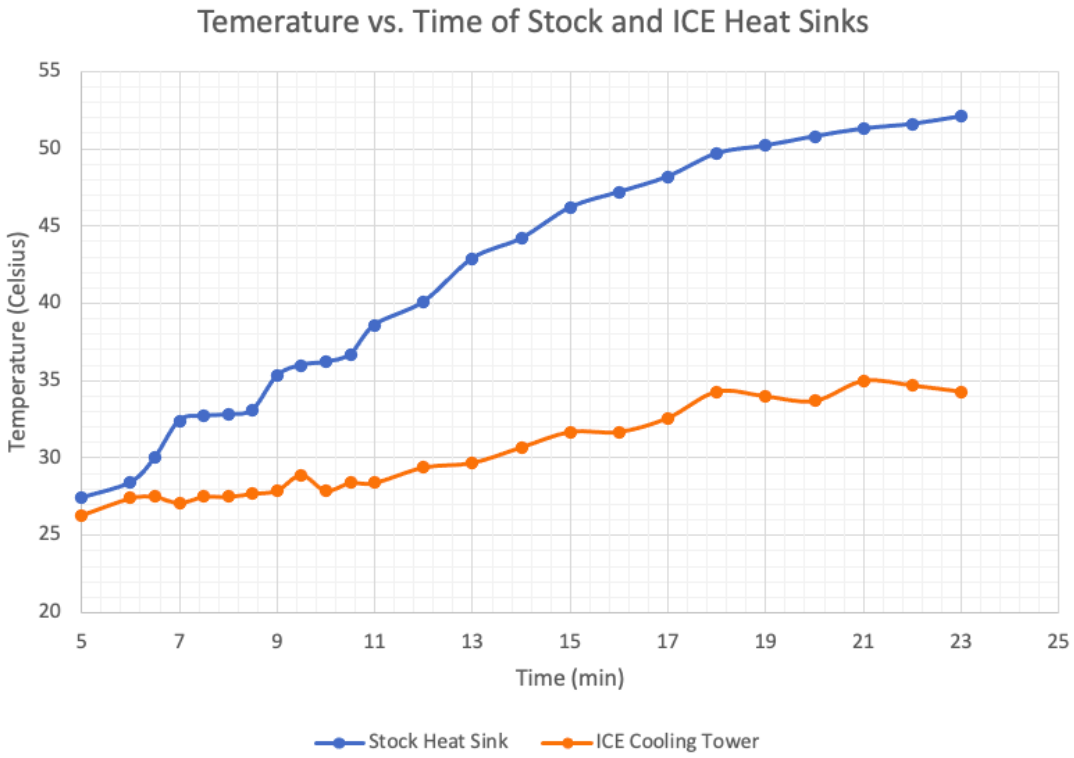


Figure 19. Comparison of the Stock and ICE heatsinks' performance with time

Prototype Test Results

Technical Specifications

Shown below are the technical specifications that were created in the fall semester. They were created to meet the four main customer needs that were identified. Sub-items are the technical requirements while the main items are the needs expressed by therapists.

1. Ease of use and speed of operation for therapists
    1.1. Mean time for new therapists to run Cervical Position Error (CPE) test shall not exceed 10 minutes (double standard CPE time).
    1.2. Proportion of therapists able to make diagnosis exclusively using the device and within 40s of completing device cycle shall exceed 50%
2. Comfort and security for patients
    2.1. Hardware shall not significantly impede movements across an 80 degree Range of Motion (RoM) of the neck about all rotational axes.
    2.2. Weight of any device attached to the patient will be less than 10 lbs.
3. Clarity and accuracy of provided information

3.1. Device shall measure angular head position across an ideal 80° and acceptable 60° RoM in either direction with error of less than 5%.
3.2. Device shall measure angular velocity up to an ideal 60° per second and an acceptable 50° per second with an error of less than 5%.
3.3. Hardware (if present) position shall shift less than 5° on the patient's head during 60° per second rotational speed.
3.4. Data shall be saved to file while immediately displaying therapist-defined output

4. Compliance
4.1. Device shall not require Personal Protective Equipment (PPE) for operation
4.2. Shall not emit sound louder than 60 dB
4.3. Any data transmission shall comply with relevant FCC and medical device regulations

	Test				
Tech Spec	1	2	3	4	5
1.1	x				
1.2		x			
3.1			x		
3.2			x		
3.4				x	
4.2					x

Figure 20. Chart corresponding technical specs to tests

Test 1

Test 1 purpose: Verify that the program can perform on test subjects in an average of under 10 minutes.

Tech Spec: 1.1

Test Method: Six Harding Physical Therapy students volunteered to help test the device. After they were briefly taught how to use the device, they used it to test either other PT students or the engineers. The time taken to use the equipment was recorded from the moment they started up the program to the moment a result was displayed.

Equipment and Environment: iPhone stopwatch and engineering building 306

Measurement System and Accuracy: iPhone accuracy is 0.01 sec, human reaction time is 0.15 sec.

Results:

TEST 1						
TRIAL	1	2	3	4	5	6
	34.68	34.19	31.55	33.9	32.11	23.13
	31.52	29.32	29.79	20.07	24.35	22.65
	30	33	23	25	17	15
	36	31	22	22	26	19
	40	28	26	23	21	20
	22	28	23	21	23	23
	29	24	24	21	20	20
	40	22	20	20	23	20
	56	28	21	26	24	29
	27	25	20	22	23	22
	45	50	34	40	25	42
	50	48	30	22	26	27
AVERAGE	36.76667	31.70917	25.36167	24.66417	23.705	23.565

Figure 21. Test 1 results

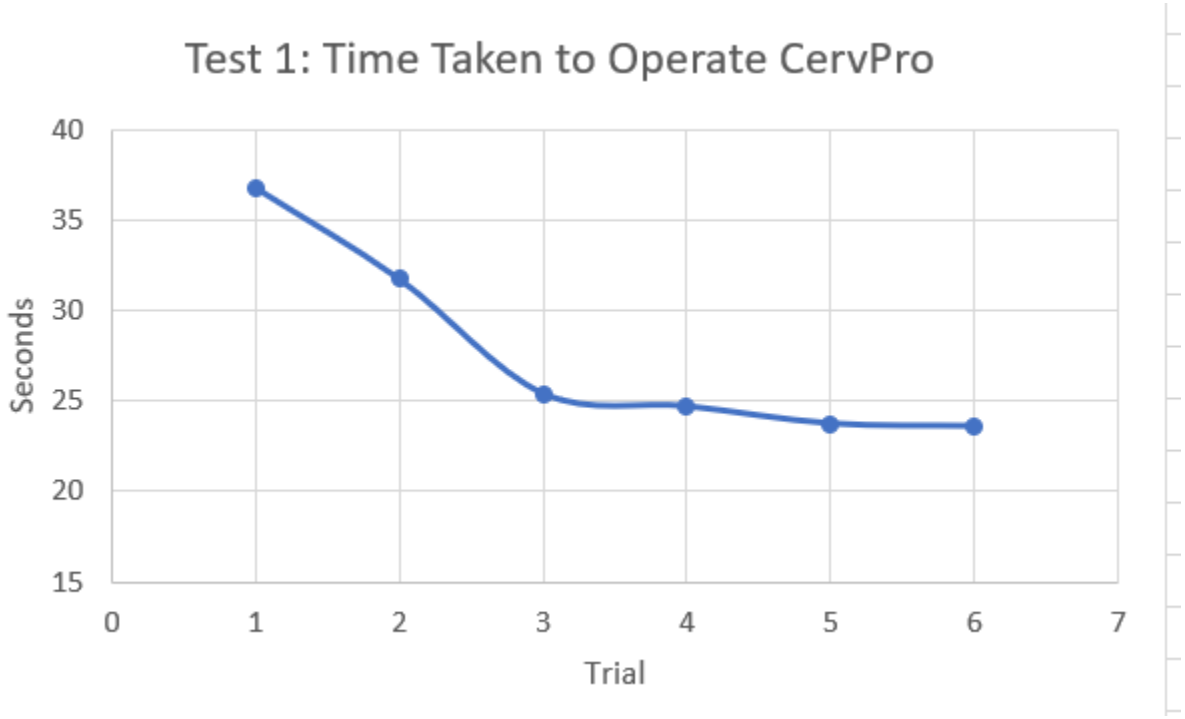


Figure 22. Graphical representation of test 1 results

As can be seen from the data, the average time was far below 10 minutes, and the time taken to operate decreased as the operator became more experienced with the device. The device performed very well in this test.

Test 2

Test 2 purpose: Verify that physical therapists find the device easy to use, and verify that at least 50% of therapists are able to make a diagnosis exclusively using the device within 40 seconds.

Tech Spec: 1.2

Test Method: Physical Therapy students and the director of the Physical Therapy program at Harding were requested to be part of the testing plan. The students had different skin tones and shapes, including dark tones. The complete program was tested and time recorded from the results displayed to diagnosis created. The Physical Therapy students used the guide shown below to operate the device and diagnose (steps 1-8).

1. Measure the distance from the eye of the camera to the head position to be 90 cm.
2. Make the patient sit on the chair facing the camera. Click the live video on and set the starting reference by instructing the patient to remain still on one position facing the camera.
3. Click the start button and let the patient move their head to the right and bring it back to its original starting position. Then click the stop button.
4. Record the angular difference and the angular speed in table 1.
5. Close the results window and restart it again.
6. Perform the test three times while recording the results.
7. Repeat step 2 and 3 above. But this time, let the patient move their head to the left.
8. Record the angular difference and angular speed in table 2.

The testing was conducted with six PT students and the head of the Harding PT program. Simultaneously, a product designer was recording time in each trial. Then the average results were calculated.

Equipment and Environment: iPhone stopwatch, engineering building and harding PT building.

Measurement System and Accuracy: iPhone accuracy is 0.01 sec, human reaction time is 0.15 sec.

Results:

	TEST 2			
	P.T. Trial	Rating	Useful in the field?	Time Taken
	1	5	Yes	5.6
	2	5	Yes	3.2
	3	5	Yes	1.7
	4	4	Yes	3.8
	5	4	Yes	2.2
	6	4.7	Yes	2.4
Average		4.617		3.15

Figure 23. Test 2 results

The Physical Therapy students were able to make diagnoses in far less than 40 seconds, showing an average of 3.15 seconds. Also, on a scale of 1-5, they rated the device at 4.617 on average as being easy to use. The device performed very well on test 2.

Test 3

Purpose: Verify that program can measure the angular position and speed of a patient's head

Tech Spec: 3.1 and 3.2

Test Method: Test subjects were positioned 90 cm away from a white board in a predetermined position, with their gaze fixed at the white board perpendicularly. The white board had preset markings on it denoting where a certain degree field of view is in relation to the test subjects' gaze angle. The CPE test was performed while a laser pointer is fixed to the subject's head. The device was then visually verified against what was seen with the laser pointer movement across the board. Angular position was measured at 5, 10, 15, 20, and 25 degrees, and the test subject was training the laser pointer at exactly the right degree measurement to verify. For angular velocity, the patient will move their head from one F.O.V. marking to the other, while being timed by a stopwatch. The program's calculation of the angular speed will then be compared to what is externally measured with the stopwatch.

Equipment and Environment: Laser pointer, white board, tape measure, protractor, test subject. SCI306.

Measurement System and Accuracy: iPhone accuracy is 0.01 sec, slow motion iPhone video 0.00415 sec error.

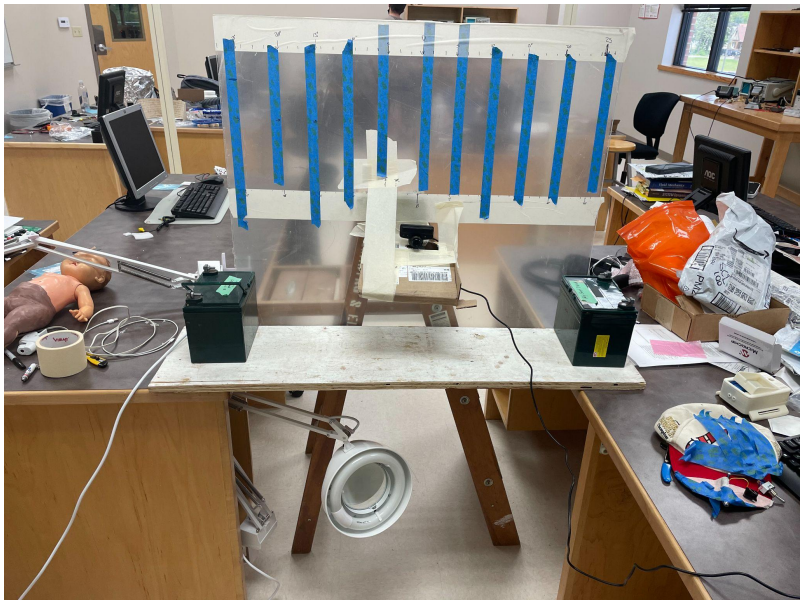


Figure 24. Photograph of test set up

Figure 25. Picture of touch screen showing program working at 50 degree angle

TEST 3		Position (deg)					
		0	5	10	15	20	25
Trial	1	0.15	4.57	9.07	10.63	22.09	27.16
	2	0.18	7.95	10.63	11.82	17.55	24.17
	3	1.88	2.7	9.8	17.55	17.55	22.09
	4	0.18	2.7	9.07	11.11	22.09	23.17
	5	0.18	2.7	9.8	11.11	20.78	24.17
	6	0.18	2.7	9.8	20.78	19.24	23.17
Average		0.4583333	3.88666667	9.695	13.8333	19.8833	23.9883
St. Dev.		0.635883	1.94123386	0.53049505	3.8991	1.90812	1.58474
		Velocity (deg/s)					
		5	10	15	20	25	30
Trial	1	1.8	9.8	13.65	20.52	23.65	23.65
	2	5.3	11.11	14.2	18.15	23.65	41.56
	3	6.41	10.26	11.82	18.15	19.6	25.73
	4	2.7	9.8	18.15	18.15	28.44	38.48
	5	4.57	9.48	12.81	21.27	23.65	22.21
	6	3.05	19.6	28.44	18.15	25.73	23.65
Average		3.9716667	11.675	16.5116667	19.065	24.12	29.2133
St. Dev.		1.5942963	3.58181402	5.68940951	1.31199	2.65668	7.76094

Figure 26. Test 3 results

This was a weaker area of the device. The device performed reasonably well on average, but the standard deviations were larger than desired. This is most likely due to user error in testing. This could be in the user stopping the watch at the right time, or the angle markings being slightly off.

Test 4

Purpose: Verify that the data is saved, and that video feedback is immediately being displayed. Tech Spec: 3.4

Test Method: Perform CPE test and observe to ensure that video feedback is immediate. Equipment and Environment: CervPro device, SCI306

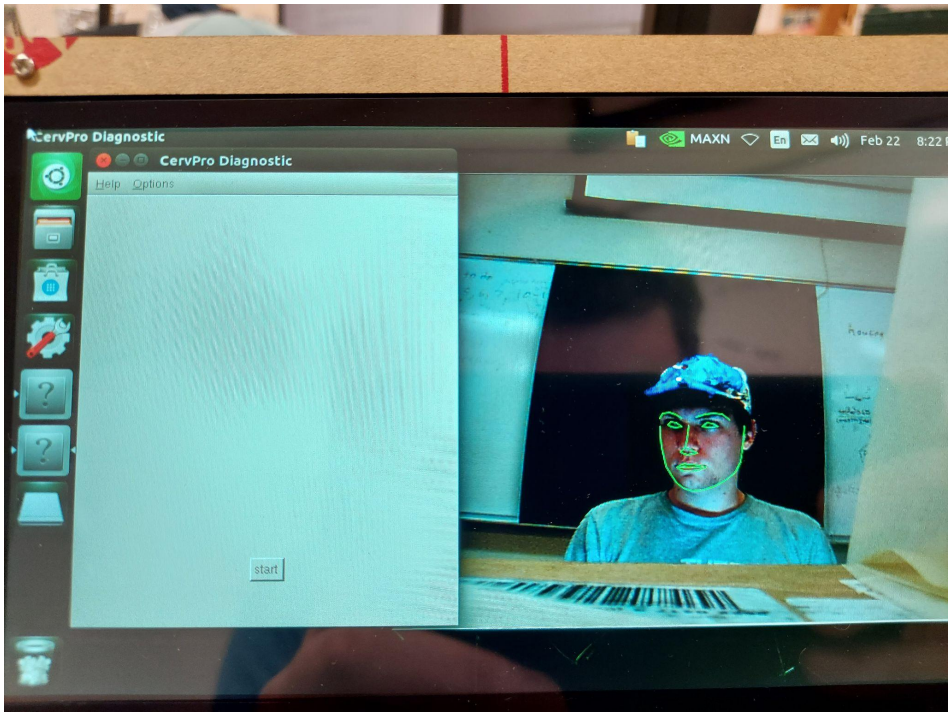


Figure 27. Device showing video feedback

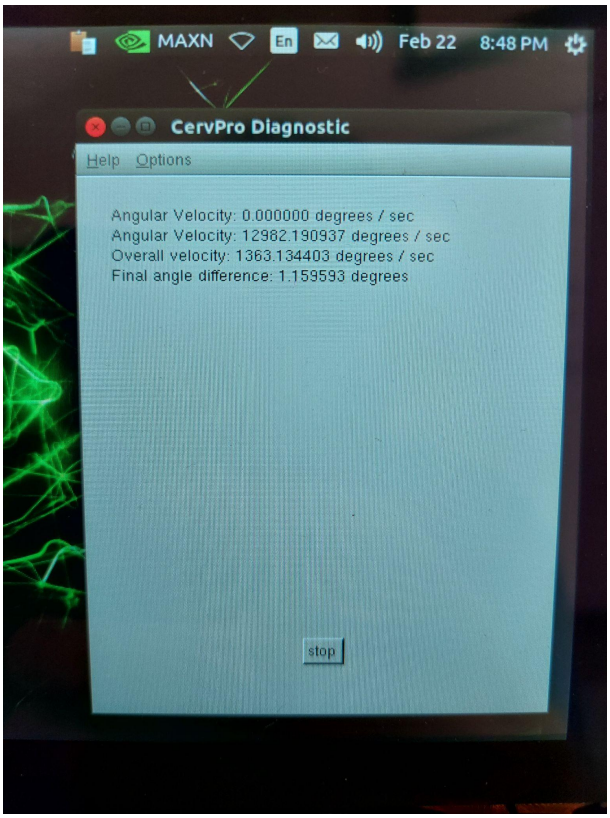


Figure 28. Data being immediately displayed once test is done

Test 5

Purpose: Ensure that the CervPro device will not be unpleasant to hear for patients and therapists.

Tech Spec: 4.2

Test Method: Record sound emitting from completed CervPro device via Datalogging Sound Level Meter to ensure that noise is not more than 60 dB

Equipment and Environment: Datalogging Sound Level Meter, SCI 306

Figure 29: Datalogging Sound Level Meter

Test Results: The sound level was found to be 43.0 dB.

Budget

	Qty	Price per each	Total Real Cost	Total Actual Cost
Power Subsystem				
Transformer	1	$24.97	$24.97	$24.97
1N4004	4	$0.00	$0.00	$0.12
2N3904	1	$0.50	$0.50	$0.50
TP31	1	$1.00	$1.00	$1.00
Resistor	3	$0.00	$0.00	$0.05
Capacitor	2	$0.00	$0.00	$0.05
LM7805	1	$0.00	$0.00	$0.75
PCB	1	$0.00	$0.00	$25.00
subtotal:			$26.47	$52.44
Camera Subsystem				
Camera	1	$37.29	$37.29	$37.29
subtotal:			$37.29	$37.29
Microprocessor Subsystem				
Microprocessor (main)	1	$100.00	$0.00	$100.00
HDMI Screen*	1	$60.00	$0.00	$60.00
ICE cooling tower	1	$24.90	$0.00	$27.90
64 GB microSD card	1	$15.00	$0.00	$15.00
subtotal:			$0.00	$202.90
Enclosure				
Sk-19	1	$49.38	$49.38	$49.38
Sk-19	1	$20.47	$20.47	$20.47
subtotal:			$0.00	$69.85
Total				$362.48
* Note: Bluetooth microprocessor was replaced with a comparible price HDMI enabled screen				

Figure 30. Updated CervPro Budget

References

1. https://towardsdatascience.com/detecting-face-features-with-python-30385aee4a8e
2. Fund, “Computer Vision on the GPU with OpenCV - Nvidia,” Computer Vision on the GPU with OpenCV. [Online]. Available: http://developer.download.nvidia.com/GTC/PDF/1085_Fung.pdf

Appendix

Appendix A: Source Code

```
/// <summery>
/// <para>This is the CervPro cervical proprioception test.</para>
/// <para>The patient sits down 90 cm from the camera and will perform
/// standard CPE test. The camera will track the patient's movement with
/// facial recognition. The difference from the starting to ending positions
/// will then be reported as an angle in degrees.</para>
/// <para>A GUI is included for the physical therapist's ease of use. There
/// are two main windows: one that shows the facial tracking in process, and
/// one that shows the points of the face on that frame (while running the test)
/// and then the final angle difference at the end. the second window has
/// a start/stop button for the test as well as a menu. The menu includes
/// instructions on how to use the machine, and options for modifying/improving
/// the test.</para>
/// </summery>
#include <dlib/opencv.h>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/videoio.hpp>
#include <dlib/image_processing/frontal_face_detector.h>
#include <dlib/image_processing/render_face_detections.h>
#include <dlib/image_processing.h>
#include <dlib/gui_widgets.h>
#include <dlib/threads.h>
#include <vector>
#include <list>
#include <fstream>
#include <iostream>
#include <sstream>
#include <cmath>
#include <iterator>
#include <time.h>

#ifndef NULL
#define NULL 0
#endif

using namespace dlib;
using namespace std;

// based on hat sizes for adults, values in cm
const std::vector<double> circumference_male = {55, 58, 60, 62, 64};
const std::vector<double> circumference_female = {55, 57, 59, 61, 63};
const double middle_circumference = 59;
```

```
const long double PI = atan(1) * 4;

const unsigned short FRAMES_PER_VELOCITY_CALC = 15;

const double ANGLE_CORRECTION = 3.50711;

const string FILE_NAME = "output_log.txt";
const string SAVE_PATH = "~/Desktop";

void Set_Circumference(double c);

/// <summery>
/// This class defines the window that will be used to display results.
/// </summery>
class my_window : public drawable_window
{

private:
 label output;
 button btn;
 menu_bar mbar;
 string output_text_copy;
 bool log;
 size_t frames;

 /// <summery>
 /// This produces a message box from pressing the Instruction menu option.
 /// </summery>
 void ShowInstructions() {
  using namespace message_box_helper;
  string str = "";
  str += "Thank you for choosing CervPro.\n\r";
  str += "The video feed is a live feed from the camera with an overlay\r\n ";
  str += "showing if a face has been detected. This overlay shows that\r\n";
  str += "the program is able to track the face, and by extension\r\n";
  str += "perform the needed calculations.\n\r";
  str += "The text shows what values are being observed by the camera\r\n";
  str += " and the results of the performed calculations of the movement\r\n of the head.\n\r";
  str += "The below button is used to start and stop the program. After\r\n";
  str += " pressing the button and causing the program to start, readings\r\n";
  str += " will begin being displayed. When the button is pressed again\r\n";
  str += " the program will make all final calculations and show how far\r\n";
  str += " off the patient's head was from where he/she started.";
  box_win *win = new box_win("Instructions", str);
  win->set_size(410, 300);
 }

 void ShowOptions() {
```

```
   options_window temp;
   temp.wait_until_closed();
  }

  void Toggle_Log() { log = !log; }

  void BtnPress() {
   if (btn_pressed) { // end test }
   else { // begin test
    btn.set_name("stop");
   }
   btn_pressed = !btn_pressed;
  }

public:
  bool btn_pressed;
  double circumference;

  /// <sumery>
  /// This creates the data window that is 400 x 500.
  /// It has a button, an output lable, and a menu bar with one item.
  /// </sumery>
  my_window() : output(*this), btn(*this), mbar(*this), log(true), frames(0),
                circumference(middle_circumference)
  {
   //button setup
   btn.set_pos(/*x*/ 175, 410 /*y*/);
   btn.set_name("start");
   btn_pressed = false;
   btn.set_click_handler(*this, &my_window::BtnPress);

   //label setup
   output.set_pos(btn.left() - 150, btn.bottom() - 400);
   output_text_copy = "";

   //menu bar setup
   mbar.set_number_of_menus(2);
   mbar.set_menu_name(0, "Help", 'H');
   mbar.menu(0).add_menu_item(menu_item_text("Instructions", *this,
&my_window::ShowInstructions, 'I'));
   mbar.set_menu_name(1, "Options", 'O');
   mbar.menu(1).add_menu_item(menu_item_text("Toggle Log", *this, &my_window::Toggle_Log, 'L'));

   ofstream fout(FILE_NAME, ofstream::trunc);
   fout << "\n\r";
   fout.close();

   //Window setup
```

```
  set_title("CervPro Diagnostic");
  set_size(380, 480);
  show();
}

/// <summery>
/// When a window is destroyed close it.
/// </summery>
~my_window() { close_window(); }

void update_log(const std::list<point> &points) {
  ++frames;
  string temp = "";
  size_t counter = 0;
  for (point p : points)
  {
    ++counter;
    if (counter % 5 == 0) temp += "\n\r";
    temp += "\t";
    temp += "(" + to_string(p.x()) + ", " + to_string(p.y()) + ") ";
  }
  //output.set_text(output_text_copy);
  if (log)
  {
    ofstream fout(FILE_NAME, ofstream::app);
    fout << temp << "\n\r";
    fout.close();
  }
}

void update_text(const string &update) {
  if (frames % 50 == 0) output_text_copy = "";
  output_text_copy += update;
  output.set_text(output_text_copy);
  if (log) {
    ofstream fout(FILE_NAME, ofstream::app);
    fout << output_text_copy << "\n\r";
    fout.close();
  }
}

void update_log(const string &update) {
  if (log) {
    ofstream fout(FILE_NAME, ofstream::app);
    fout << output_text_copy << "\n\r";
    fout.close();
  }
}
```

```
  /// <summery>
  /// This method is used for finding the best way to calculate the best
  /// way to find the ending difference.
  /// </summery>
  void update_text(const std::vector<double> final_values) {
    output_text_copy = "";
    output.set_text(output_text_copy);
    for (double d : final_values)
    {
      string str = "Final angle difference: " + to_string(d) + "\r\n";
      update_text(str);
    }
  }
};

std::list<std::list<point>> stored_faces;
my_window data_win;

void Set_Circumference(double c) { data_win.circumference = c; }

double radius(double circumference) { return circumference / (2 * PI); }

double Euclidean_Distance(point p1, point p2) { return sqrt(pow(p1.x() - p2.x(), 2) + pow(p1.y() - p2.y(),
2)); }

double Euclidean_Distance(point p) { return sqrt(pow(p.x(), 2) + pow(p.y(), 2)); }

double pixel_to_real(int dist) { return (90 * dist * 0.635) / (0.413774 * 480); }

double angular_correction(double angle) {
  return 0.0000020278 * pow(angle, 7) - 0.0001340304 * pow(angle, 6) + 0.0032900011 * pow(angle, 5) -
0.0356139393 * pow(angle, 4) + 0.143281671 * pow(angle, 3) + 0.0795877567 * pow(angle, 2) +
0.0272896557 * angle + 0.0076780359;
  return angle;
}

double final_calculation(const size_t &framecount) {
  auto begining = stored_faces.begin()->begin();
  auto ending = stored_faces.rbegin()->begin();
  advance(begining, 33);
  advance(ending, 33);
  // returns in radians, so convert to degrees
  double angle = atan(pixel_to_real(Euclidean_Distance(*begining, *ending)) / (/*h*/
radius(data_win.circumference))) * 180 / PI;
  return abs(angular_correction((angle - ANGLE_CORRECTION) * 0.6));
}
```

```
double avg(std::vector<double> container) {
  double result = 0;
  std::vector<double>::size_type i = 0;
  for (i; i < container.size(); ++i)
  {
    result += container[i];
  }
  return result / i;
}

/// <summery>
/// This function calculates the final difference between
/// </summery>
std::vector<double> final_calculation() {
  std::vector<double> result;
  size_t counter = 0;
  double distances = 0;
  for (auto i = stored_faces.begin()->begin(), j = stored_faces.rbegin()->begin(); counter <
stored_faces.begin()->size(); ++i, ++j, ++counter) {
    double distance = Euclidean_Distance(*i, *j);
    result.push_back(abs(atan(pixel_to_real(distance) / (radius(data_win.circumference) / 2)) * 180 / PI) -
ANGLE_CORRECTION);
  }
  result.push_back(avg(result));
  result.push_back(atan(pixel_to_real(distances / stored_faces.size()) / (radius(data_win.circumference) /
2)) * 180 / PI);
  return result;
}

double angular_velocity(int start_frame, time_t time_passed) {
  auto temp = stored_faces.begin();
  advance(temp, start_frame);
  auto begining = temp->begin();
  auto end = stored_faces.rbegin()->begin();
  auto old = stored_faces.begin()->begin();
  advance(end, 33);
  advance(old, 33);
  double theta = atan(pixel_to_real(Euclidean_Distance(*begining, *end)) / (/*h*/
radius(data_win.circumference))) * 180 / PI;
  double rho = atan(pixel_to_real(Euclidean_Distance(*old, *begining)) / (/*h*/
radius(data_win.circumference))) * 180 / PI;
  theta = angular_correction(abs(theta - ANGLE_CORRECTION) * 0.6);
  rho = angular_correction(abs(rho - ANGLE_CORRECTION) * 0.6);
  return abs((theta - rho) / time_passed);
}

/// <summery>
/// Calculate the velocity from start to end over the time_passed.
```

```
/// </summery>
/// <return>The angular velocity</return>
double angular_velocity(int start, int end, time_t time_passed) {
  auto start_itr = stored_faces.begin();
  auto end_itr = stored_faces.begin();
  advance(start_itr, start);
  advance(end_itr, end);
  auto start_point = start_itr->begin();
  auto end_point = end_itr->end();
  advance(start_point, 33);
  advance(end_point, 33);
  double theta = atan(pixel_to_real(Euclidean_Distance(*start_point, *end_point)) /
radius(data_win.circumference)) * 180 / PI;
  theta = angular_correction(abs(theta - ANGLE_CORRECTION) * 0.6) * 2;
  return abs(theta) / time_passed;
}

/// <summery>
/// Find the point with the largest value
/// </summery>
/// <return>Largest point value</return>
int max_point() {
  int result = -1;
  double max_val = 0;
  size_t counter = 0;

  for (auto itr : stored_faces) {
    auto temp = itr.begin();
    advance(temp, 33);
    double d_temp = Euclidean_Distance(*temp);
    if (d_temp > max_val) {
      max_val = d_temp;
      result = counter;
    }
    ++counter;
  }
  return result;
}

int min_point()
{
  int result = -1;
  double min_val = INFINITY;
  size_t counter = 0;

  for (auto itr : stored_faces) {
    auto temp = itr.begin();
    advance(temp, 33);
```

```
    double d_temp = Euclidean_Distance(*temp);
    if (d_temp < min_val) {
      min_val = d_temp;
      result = counter;
    }
    ++counter;
  }

  return result;
}

int main() {
  try {
    cv::VideoCapture cap(0, cv::CAP_V4L);
    thread_pool &global_pool = default_thread_pool();

    if (!cap.isOpened()) return 1;

    image_window win;

    // Load face detection and pose estimation models.
    frontal_face_detector detector = get_frontal_face_detector();
    shape_predictor pose_model;
    deserialize("shape_predictor_68_face_landmarks.dat") >> pose_model;

    size_t frame_count = 0;
    size_t frames_checked = frame_count;
    unsigned short btn_presses = 0;

    time_t start, end, true_start, true_end;

    start = time(NULL);

    // Grab and process frames until the main window is closed by the user.
    while (!win.is_closed()) {
      // Grab a frame
      cv::Mat temp;
      if (!cap.read(temp)) break;
      cv_image<bgr_pixel> cimg(temp);

      // Detect faces
      std::vector<rectangle> faces = detector(cimg);
      // Find the pose of each face.
      std::list<full_object_detection> shapes;
      for (unsigned long i = 0; i < faces.size(); ++i)
      shapes.push_back(pose_model(cimg, faces[i]));

      if (data_win.btn_pressed && !shapes.empty()) {
```

```
    if (frame_count == 0) true_start = time(NULL);
    /// <summery>
    /// This function copies the values from temp and stores them in stored_faces.
    /// </summery>
    // lambda example
    global_pool.add_task_by_value([=, &shapes, frame_count]() {
      std::list<point> temp2;
      auto temp3 = shapes.begin();
      //advance(temp3, frame_count);
      for (size_t i = 0; i < temp3->num_parts(); ++i) temp2.push_back(temp3->part(i));
      stored_faces.push_back(temp2);
      data_win.update_log("\r\nFrame count: " + to_string(frame_count));
      data_win.update_log(temp2);
    });

    if (frame_count % FRAMES_PER_VELOCITY_CALC == 0 && frame_count != 0) {
      // angular difference
      end = time(NULL);
      time_t time_diff = difftime(end, start);

      data_win.update_text("\n\rAngular Velocity: " + to_string(angular_velocity(frames_checked,
time_diff)) + " degrees / sec");
      frames_checked = frame_count; // update the counter
      start = time(NULL);
    }

    ++frame_count;

    // only increment it once because the button was only pressed once
    if (btn_presses == 0) ++btn_presses;

    if (btn_presses == 1 && !data_win.btn_pressed) {
      true_end = time(NULL);
      ++btn_presses;
      win.close_window();

      global_pool.wait_for_all_tasks();

      // overall velocity
      double max_angle_v = angular_velocity(min_point(), max_point(), difftime(true_end, true_start));
      string temp_string = "\n\rOverall velocity: " + to_string(max_angle_v) + " degrees / sec";
      data_win.update_text(temp_string);

      // final calculations
      double angle_dif = final_calculation(frame_count);
      string final_calc_str = "\n\rFinal angle difference: " + to_string(angle_dif) + " degrees";
      data_win.update_text(final_calc_str);
      //data_win.update_text(final_calculation());
```

```
      break;
     }

     // Display it all on the screen
     win.clear_overlay();
     win.set_image(cimg);
     win.add_overlay(render_face_detections(std::vector<full_object_detection>(shapes.begin(),
     shapes.end())));
    }
    stringstream ss;
    ss << "cp -f " << FILE_NAME << " " << SAVE_PATH;
    system(ss.str().c_str());
    ss.ignore(255);
    ss << "rm " << FILE_NAME;
    system(ss.str().c_str());
    data_win.wait_until_closed();
  }
  catch (serialization_error &e) { return -5; }
  catch (exception &e) { return -6; }
}
```

Appendix B: Program Flow Chart

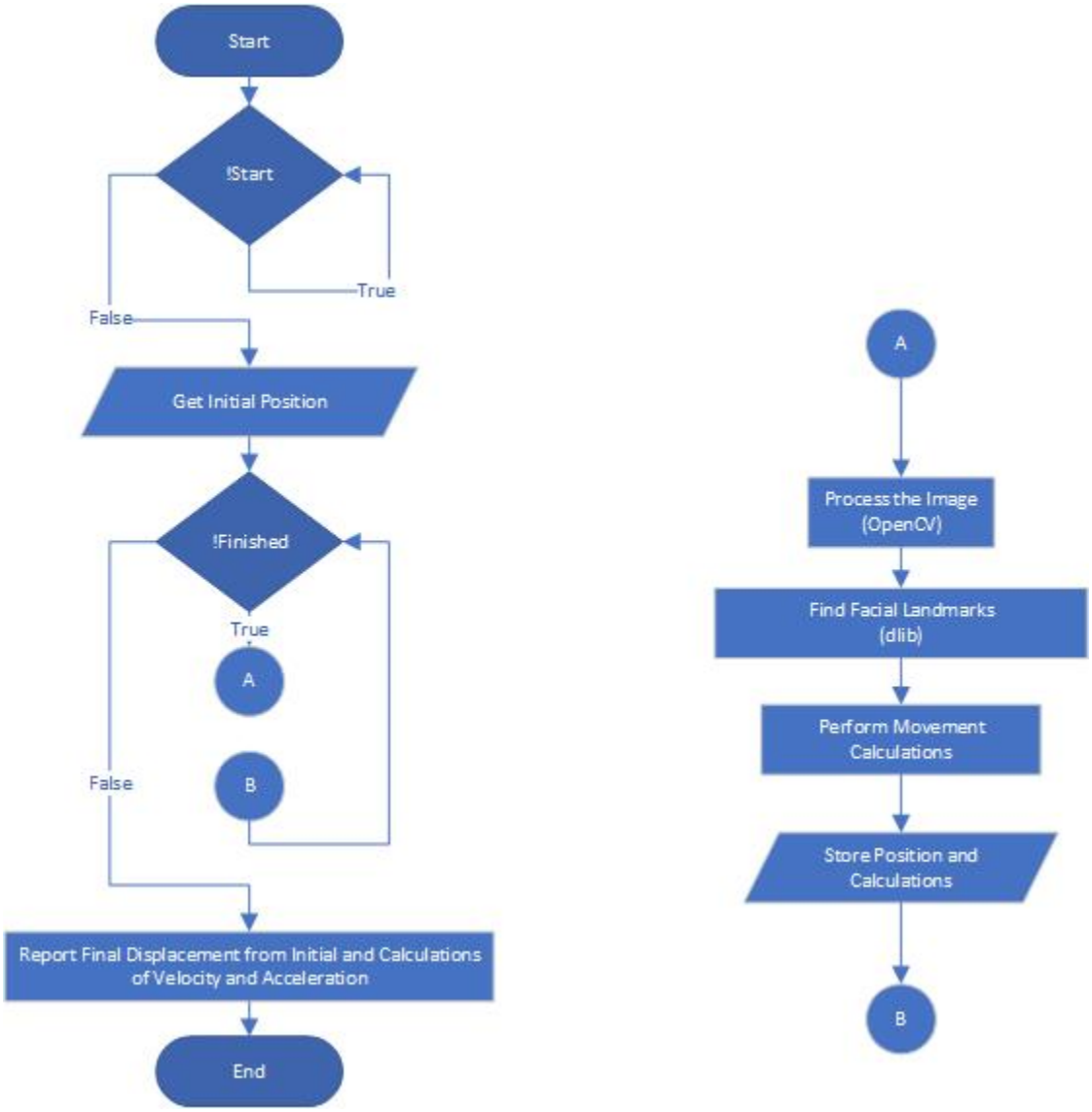


Figure A1. CervPro flow chart.

Appendix C: State Chart

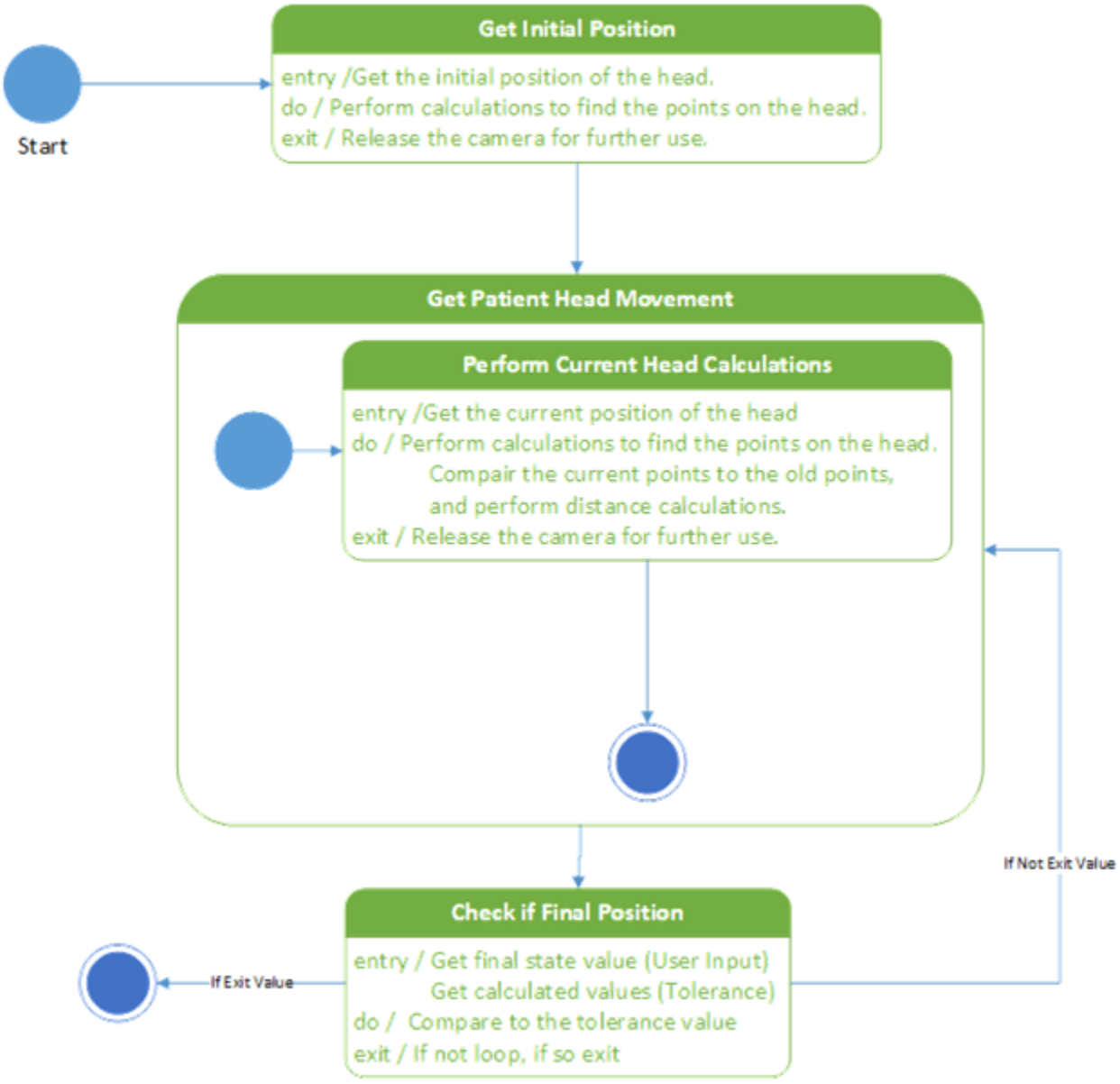


Figure A2. CervPro state chart.

Appendix D: GPU Facial Detection Speed Test

Below is the results of a GPU based speed test for facial recognition conducted by Nvidia. This data is a few years old but used to see a reference. Nvidia tested multiple different image qualities with 640x480 being the quality of our camera.

Figure A3. Nvidia facial detection speed test[2].

Appendix E: RASIC

	Role\Person	Nathan	Ethan	George	Keneth
Mechanical	Housing	I	R	I	I
	Cooling	I	S	S	R
	Kinesiology	S	R	I	I
Computer	Microprocessor	I		R	
	Programming	I	I	R	
Electrical	Power	R	I	I	
	ESD	R	S	I	
Documentation	A3	S	R	S	I
	Reports	S	R	S	I
Misc	Budget	S	R	S	I
	Compliance	R	S		

Figure A4. R.A.S.I.C