UC SANTA CRUZ
Baskin Engineering

Capstone Project

Eye Tracking Autism for Mobile

John-Michael Burke, Nate Cancio, Mike Madsen

ETAM

CITRIS

Abstract

The Eye Tracking Autism for Mobile (ETAM) project was done to develop a proof of concept mobile system which uses eye-gaze tracking to observe the user's preference of certain imagery to aid in the early detection of autism spectrum disorder in children. The concept follows research done by Dr. Karen Pierce which showed how the preference of geometric patterns can be used as a method of detecting autism in children as young as 14 months old. She used a premade stationary eye-gaze tracker to conduct her research. So the ETAM team sought to develop a mobile version of her autism examination. In order to do this we needed to develop a gaze estimation algorithm based around the constraints of our hardware. Although there has been significant research done, the results of this test should not be interpreted as final diagnosis. Instead we consider this examination as an instigator for seeking a more thorough diagnosis by a medical professional.

Approach

Our goal with the project was to create an application for a mobile system to perform the autism tests. In order to create this application we used a pre-existing pupil tracker which implemented the Timm and Barth gradient pupil localization algorithm that included a kalman filter for smoothing out the face detection. Then we needed to design and develop a method of converting the pupil location data into a gaze position on the screen. So we use an algorithm to track the corners of the eyes which are used as a reference point. Then we compute an averaged pupil's distance away from the center of the reference rectangle in order to determine where the user is looking on the screen or to see if they are looking at the screen at all. The gaze data and timestamps are recorded while the user watches a special video which will eventually be used to attempt the experimental diagnosis.

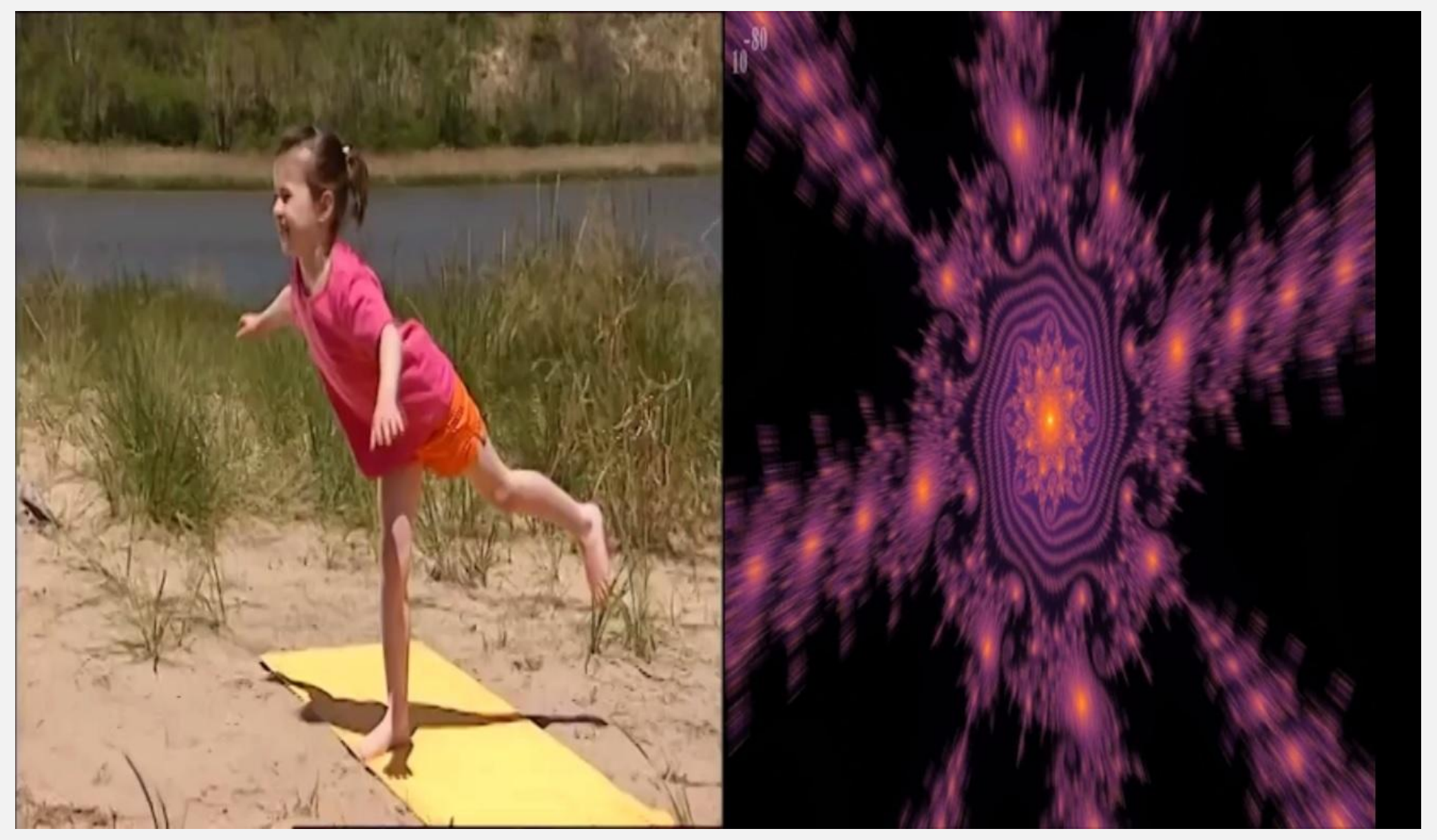


Figure 1: Social and geometric video used in our test

In Dr. Karen Pierce's research she would have the child watch a special video which was divided into two sides, one side would display a video of children playing and doing yoga and the other side displayed fractals and geometric patterns. We created our own special video which is an attempt at replicating the unique videos she used. Observing the computed data we use the position and duration of the user's gaze to perform the experimental diagnosis, which we quantify into a potential risk percentage. We took this test and embedded it into a GUI database that allows a medical professional to store and lookup patient information, run a new test, or open a previously stored test to watch the test video with the users gaze overlaying it. Ideally we would implement this application onto a phone or tablet which would have an inhouse touchscreen and keyboard but for being a student funded project we pulled off a decent proof-of-concept system.

Overview

The System The Jetson features a quad-core ARM Cortex A15 CPU/GPU processor, the Tegra K1, which is the first mobile processor to support CUDA. The CUDA support allows us to use OpenCV's GPU module to accelerate the speed of various functions such as our face detection. On the Jetson we worked with Nvidia's in house Ubuntu operating system Linux4Tegra (L4T) R21.3 and loaded the a latest version of a custom kernel, the Grinch, for added driver support and some useful applications.

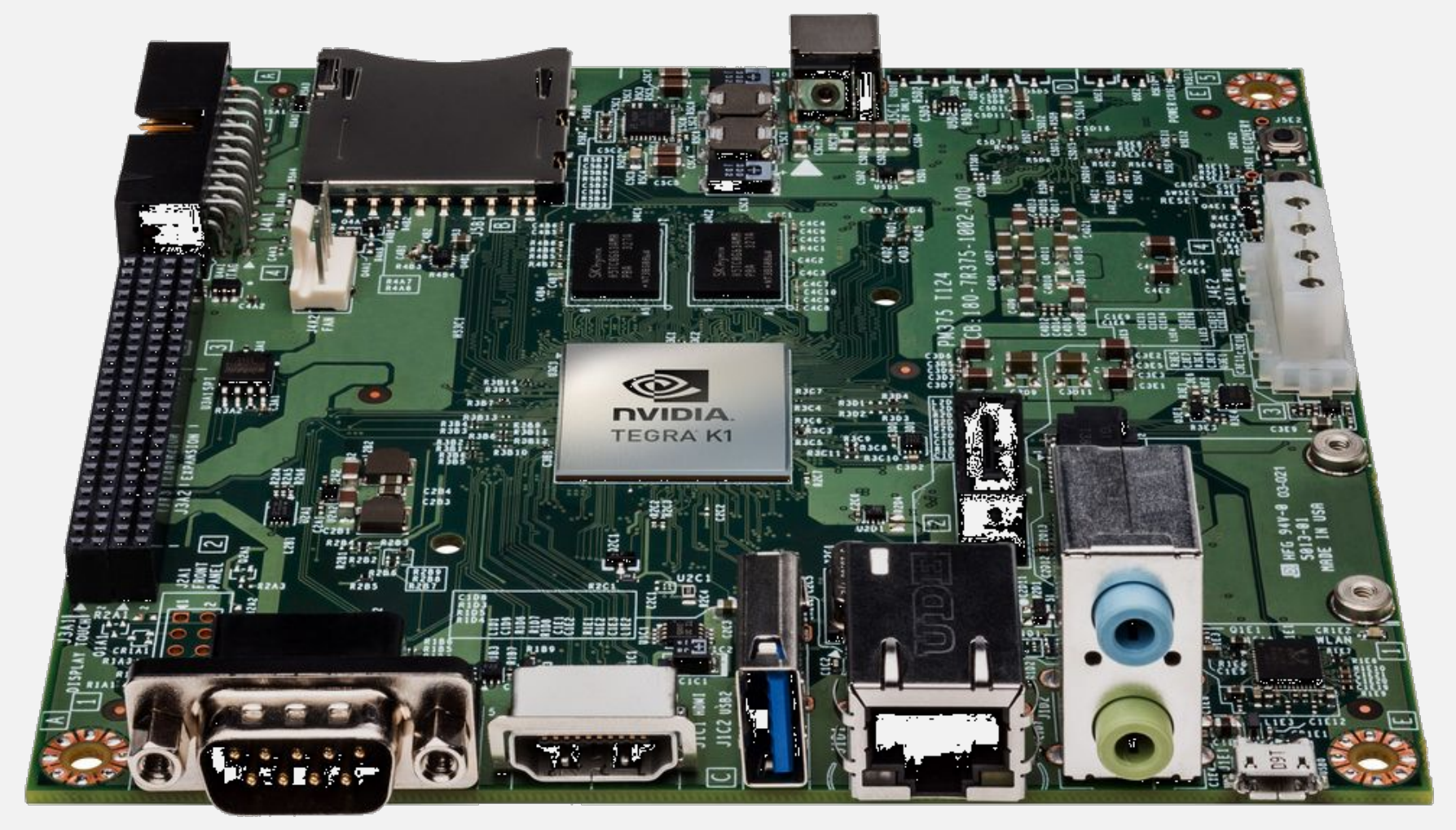


Figure 2: Nvidia Jetson Tegra K-1 dev. kit

The Software consisted of a haar classifier for face detection, a gradient-based pupil tracker, a subspace constrained mean-shifts facial landmark detector, a graphical user interface, and a locally contained database. For our pupil tracker, we used code written by Tristaan Hume called *eyeLike* which implements the Timm and Barth gradient pupil localisation algorithm. The version we are using as our baseline was modified by Gabriel Huang which has an addition of a kalman filter for smoothing out the detected face over time. We have optimized the code to maximize the performance running on the Jetson by using multi-threading and OpenCV's GPU module. The multithreading configuration we found to be the most beneficial is to read and display the test video on one detached thread, the two pupil center localizations running on another two separate threads, and finally the main thread that the application is run on and the camera feed is assessed. We have also made sure to set the GPU and CPU clock settings to the absolute maximum for the best performance when multithreading. We also stripped countless features from the Jetson to run the application without any background processes hampering the potential of the CPUs or GPU.

Gaze Estimation After attempting to design and implement countless approaches to this feature, we came up with a low resolution but stable solution using the a pupil tracker and a facial landmark tracker. The facial landmark tracker, FaceTracker, finds the face one time and then uses a method mean-shifting to search for changes in the position of the landmarks. Although the tracker finds up to 66 different landmark points on the face we are only using 12 of them, 6 for each eye. We use these points to construct a reference box around the corneas to observe the pupil positions with respect to the reference box centers. We account for minor differences in the position of the eyes by using a maximum averaged reference box and average pupil position. Then we quantify the difference as a percentage of the width (x axis) or the height (y axis) of the averaged reference box. Then we can easily modify ranges using these percentages in order to increase or decrease the gaze resolution. Then we use the observed precision of our gaze estimation process to specify how many different ranges of percentages we look for which can be thought of as our algorithms resolution or "how many divisions we can accurately separate our gaze estimation into." based on the amount of ranges we can quantify. Granted, there is always a limit to the resolution but we can quickly reset it based on how much noise we observe.

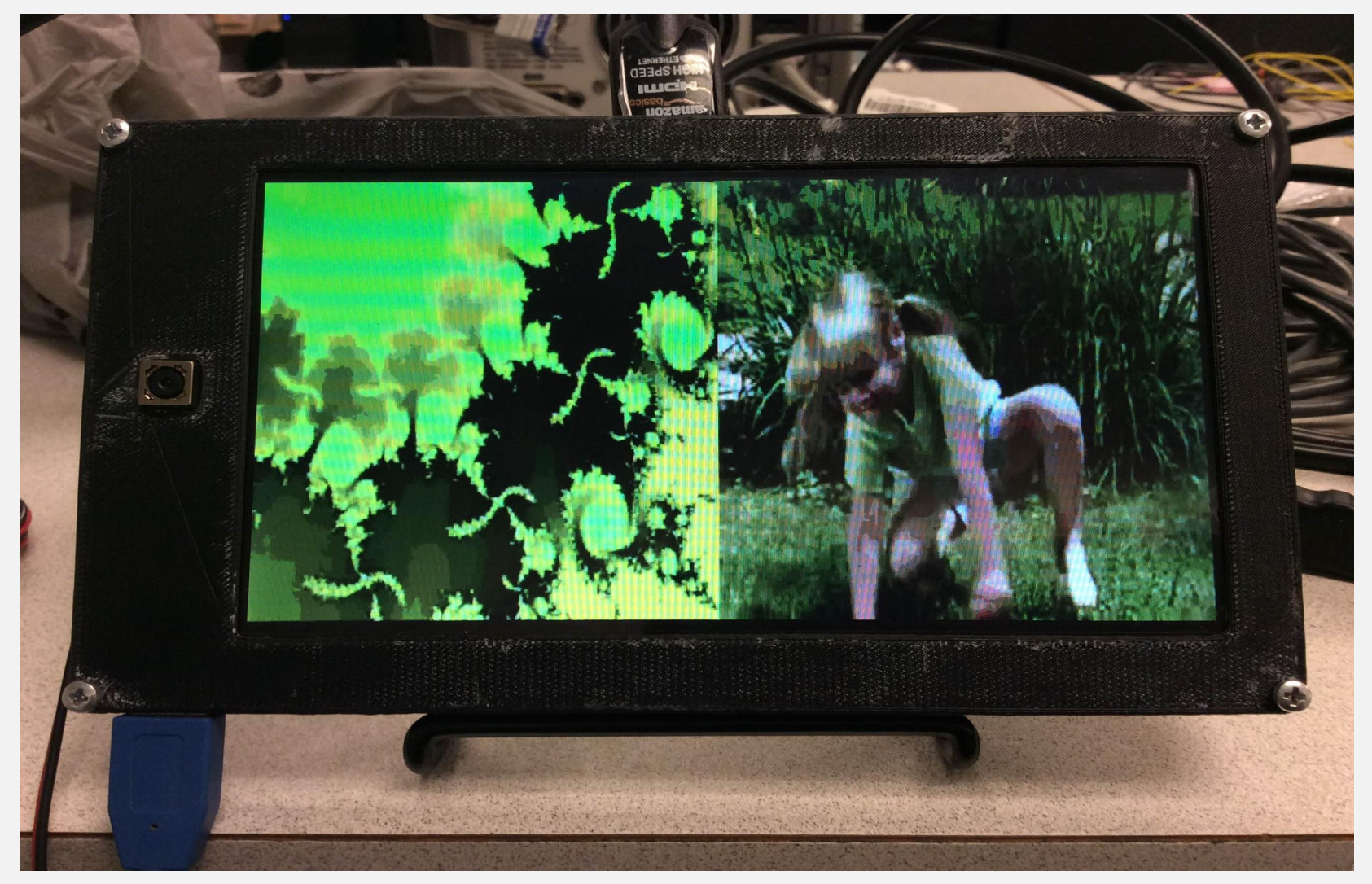


Figure 3: System running test

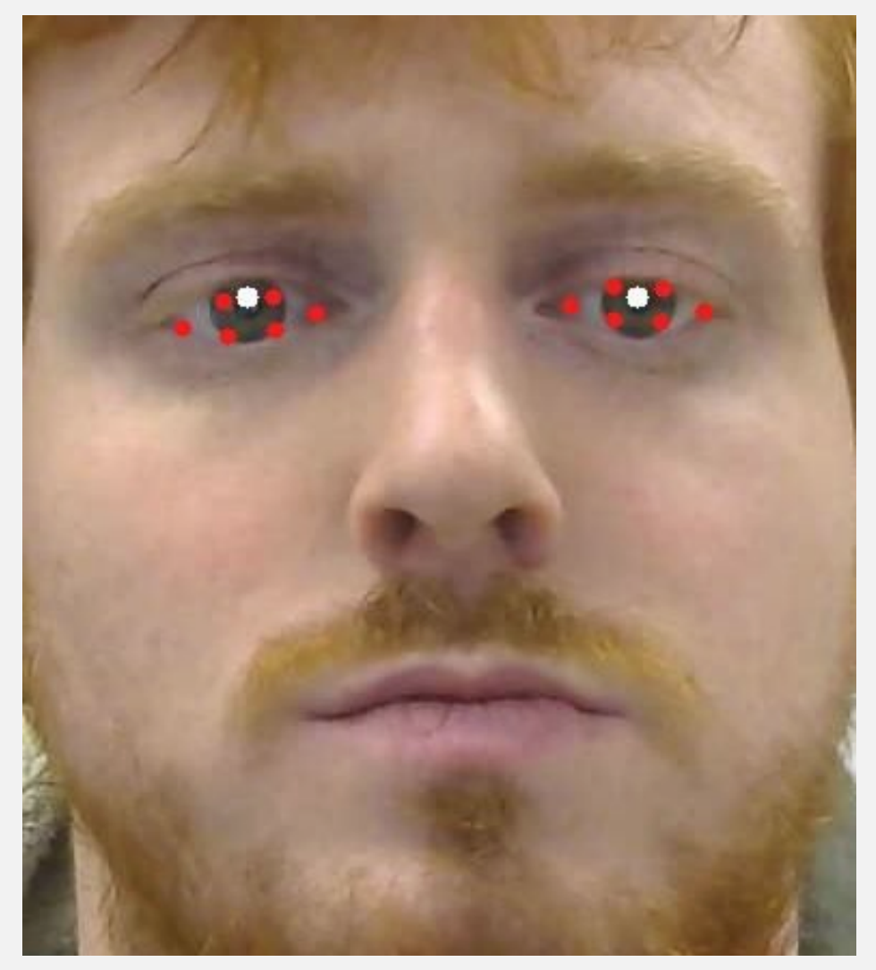


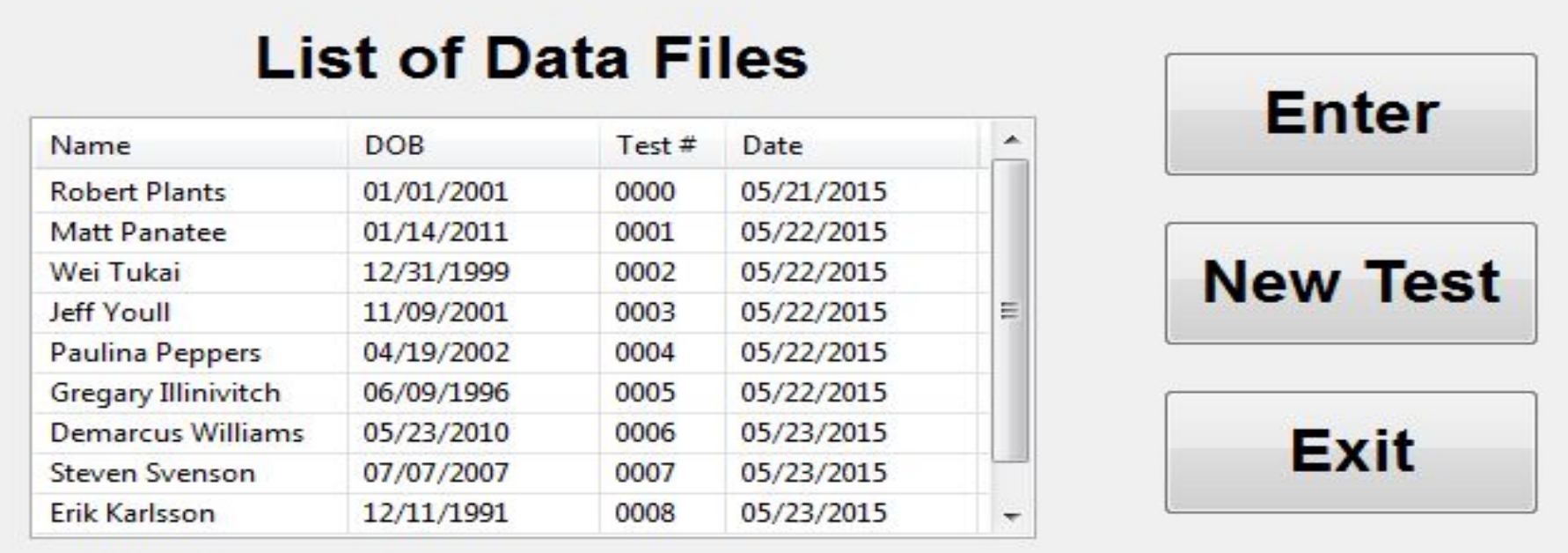
Figure 4: Example of pupil tracking algorithm

Acknowledgments

We would like to thank senior design project mentors Pat Mantey and Paul Naud for help and guidance throughout the duration of the project. We would like to acknowledge the support of Pat Mantey and CITRIS. We would also like to acknowledge the instrumental research of Dr. Karen Pierce of UC San Diego's Autism Center for Excellence. The first source we must cite is from Tristaan Hume for *eyeLike* as well as some additions made by Gabriel Huang. Finally, we heavily adapted *FaceTracker* by Jason M. Saragih. We are not using any our code, which include adaptations of these works, for commercial use, it is solely for academic purposes.

Analysis

To evaluate the performance of the gaze estimation, we record the data gathered over each run of the autism examination. The gaze estimation data is then post-processed and scored based on the duration of strong fixations as well as the preference of video stream. Dr. K ren Pierce found that when a toddler focuses on the geometric patterns for at least 69% of the duration of the video, the likelihood of that child having ASD was 100%. We store the data in a local database and have created a feature allowing the physician to watch the user's gaze overlayed on top of the specific test video. We have created an interactive graphical user interface to easily navigate through previous patient data analysis, recording examination data for new or previous patients, as well as replaying a patient's examination data superimposed on to the video they viewed. This helped us structure our application to be seamlessly embedded into a professional environment.


List of Data Files

Name	DOB	Test #	Date
Robert Plants	01/01/2001	0000	05/21/2015
Matt Panatee	01/14/2011	0001	05/22/2015
Wei Tukai	12/31/1999	0002	05/22/2015
Jeff Youll	11/09/2001	0003	05/22/2015
Paulina Peppers	04/19/2002	0004	05/22/2015
Gregary Illinivitch	06/09/1996	0005	05/22/2015
Demarcus Williams	05/23/2010	0006	05/23/2015
Steven Svenson	07/07/2007	0007	05/23/2015
Erik Karlsson	12/11/1991	0008	05/23/2015

Figure 5: GUI Patient Directory

Results

Due to liability issues and the short duration of this project, we were not able to organize a live demo in order to verify the results of our implementation of the experimental autism diagnosis. We did manage to achieve acceptable results when testing on ourselves and passersby outside our lab. As for the performance of our system, we had some notable feats for a mobile prototype. Pushing the limits on our Jetson's GPU, we were able to achieve a 40% increase in framerate for our Haarcascade face detection. In addition to this, we implemented a pre-existing Kalman filter which greatly stabilized the detections. Then we successfully added our own Kalman filter to stabilize the detected pupil positions even further. The facial landmark tracker ran at an acceptable rate even using 720p images but could have been optimized to use the GPU. By offloading some of the computation to the GPU and even multi-threading the application to run things in parallel, we were able to run the gaze estimation algorithm at roughly 5 fps with a resolution of 4 divisions and simultaneously display the examination video at the full 30 frames per second.

Conclusion

The Eye Tracking Autism for Mobile project is intended to be a proof-of-concept for a mobile version of an experimental method for the early detection of autism spectrum disorder in children. We used a powerful new mobile CPU/GPU processor to show that eye-gaze tracking on a mobile device is now possible and can be used to support computationally expensive applications such as as gaze tracking. This in turn opens the door for extremely beneficial and unique applications such as the early detection of autism. Due to unforseen obstacles, we were limited to using primarily pre-existing algorithms for the pupil and eye corner detection. However, as a post senior design goal we intend to implement our own method for eye corner detection which will be heavily integrated with GPU hardware accelerations.