## Improving the accuracy of low-quality eye tracker

**Author(s)** You, Han (游晗)

**Citation** You, H. (2018). Improving the accuracy of low-quality eye tracker (Outstanding Academic Papers by Students (OAPS), City University of Hong Kong).

**Issue Date** 2018

**URL** http://dspace.cityu.edu.hk/handle/2031/9114

**Rights** This work is protected by copyright. Reproduction or distribution of the work in any format is prohibited without written permission of the copyright owner. Access is unrestricted.
17CS002

Improving the Accuracy of Low-quality Eye Tracker

(Volume 1 of 1)
Acknowledgement

I’d like to appreciate Dr. Antoni Chan, who is the supervisor for this project, for his patience and innovative ideas throughout the past year. Although being busy, he managed to arrange weekly meeting with me to keep an eye on my project and help me find a way out when there’re problems. It is very lucky for me to have such a nice supervisor for my final year project.

In addition, I’d like to thank Ms. Vicky Chan in the Department of Media and Communication for helping me arrange the Eye Tracking Lab for many and many times, and Mr. Jim in the CS Lab for helping me on the experiments.
Abstract

Eye tracking, refers to the process of measuring the eye gaze. An eye tracker is a device for measuring the position of eye gaze. The increased accuracy and accessibility of eye-tracking technologies in recent years have made it popular in many applications such as web usability, automotive driving and advertising. Recently, there are also new eye tracking applications appearing in HCI area. For example, eye tracking can be used to help the disabled to use computer efficiently, as they can jump between different applications by moving their eye fixations. However, most traditional hardware eye trackers are inconvenient to deploy in daily life. In recent years, with the rapid advancement in deep learning, some researchers have turned to Convolutional Neural Network (CNN) to do eye tracking, in which the inputs are the images of user’s face or eyes and the output will be the predicted eye gaze coordinate.

In this project, I aim to improve an existing eye tracking model $iTracker$ from CSAIL for predicting the eye gaze. Since this model is trained for mobile phone, and I’d like to provide a solution for eye tracking in desktop, so I manage to port this model for computer. Afterwards, I use Kalman Smoother and CNN to process the output of $iTracker$ to improve the accuracy of this model so that it will be capable of handling eye tracking tasks in daily scenarios.
1. Introduction 7
  1.1 Background 7
    1.1.1 Eye Tracking Applications 7
    1.1.2 Overview of Existing Choices 8
  1.2 Objectives 9

2. Literature Review 11
  2.1 Evaluation of Existing Eye Trackers 11
    2.1.1 Hardware-based Eye Trackers 11
    2.1.2 Software-based Eye Trackers 12
  2.2 Market Analysis 13
  2.3 Problem Scope 15
    2.3.1 Failed Attempts 15
    2.3.2 Problem Statement & Solution 16

3. Preliminary Work 19
  3.1 Mapping iTracker for Computers 19
    3.1.1 Preparing Inputs 19
    3.1.2 Calibration Setup 20

4. Methodology 23
  4.1 Data Collection 23
  4.2 Data Processing 25
  4.3 Time-Series Model 26
    4.3.1 Kalman Filter vs Kalman Smoother 26
    4.3.2 Expectation-Maximization (EM) Algorithm 27
    4.3.3 Fixed-window Time-series Model 28
    4.3.4 Flexible-window Time-series Model 29
  4.4 Regression Model 30
    4.4.1 Saliency Map 30
    4.4.2 Inputs 31
    4.4.3 Model Structure 32

5. Result Analysis 34
  5.1 Results of Time-series Model 34
  5.2 Results of Regression Model 34
6. Future Improvement 35
   6.1 Retrain iTracker 35
   6.2 Use Fixation as Input 36

7. References 37

8. Appendix 39
   A. Monthly Logs 39
1. Introduction

1.1 Background

1.1.1 Eye Tracking Applications

Eye tracking technology has a long history that dates back 1800s. Nowadays, the most widely used method is video-based eye tracking. In eye tracking process, a camera would take video of the eyes. At the same time, the eye tracker uses the center of the pupil and infrared light pattern to generate a purkinje image, as shown in Figure 1. Afterwards, the eye tracker can collect information of eyes from the video, and use those information to compute the gaze direction. Mostly, the calibration process is required before eye tracking to fit different users and different working scenarios.

![Figure 1: Purkinje image](image)

Eye tracking technology has applications in different areas that range from psychology to human-computer interaction. On the one hand, analyzing eye gaze of users can provide much information for analyzing user’s behavior, which will be very useful in advertising. For example, some supermarkets are making use of eye tracking to analyze where people pay attention to. As shown in
Figure 2, the eye gaze data collected can be a good start for sales staffs to arrange the commodities to improve the promotion efficiency.

In addition, eye tracking technology can be used to improve user experience in using the internet. For instance, eye tracking can help the disabled to use computers without moving their hands. Instead, they just need to move their eyes on the screen and jump between different applications, which will definitely bring convenience to them.

1.1.2 Overview of Existing Choices

Roughly speaking, there are two kinds of existing eye trackers, hardware-based and software-based eye trackers.

Hard-based eye trackers usually have an eye tracking module for taking a video of user’s eyes, emitting infrared ray, and receiving reflected optical features. Thus, hard-based eye trackers cannot be very small. In addition, they typically require connection with computers to get electricity and transmit data.
Software-based eye trackers, which depend on the images taken by web camera to make a prediction, appeared in recent two decades. Compared to traditional hardware-based eye trackers, they do not require power support and can run as an application. However, limited by the accuracy of prediction model, the performance of previous software-based eye trackers are usually worse than hardware-based eye trackers. In recent years, many researchers have started to train CNN for predicting eye gaze, which produces a much better performance than previous SVM based software-based eye trackers and it already meets the basic requirements in daily scenario.

<table>
<thead>
<tr>
<th>Model</th>
<th>Error (cm)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Center</td>
<td>7.54</td>
<td>Simple baseline</td>
</tr>
<tr>
<td>TurkerGaze</td>
<td>4.77</td>
<td>pixel gfeatures + SVR</td>
</tr>
<tr>
<td>MPIIGaze</td>
<td>3.63</td>
<td>CNN + head pose</td>
</tr>
<tr>
<td>TabletGaze</td>
<td>3.17</td>
<td>Random forest + mHoG</td>
</tr>
<tr>
<td>AlexNet</td>
<td>3.09</td>
<td>eyes + face + fg.</td>
</tr>
<tr>
<td>iTracker</td>
<td>2.58</td>
<td>fc1 of iTracker + SVR</td>
</tr>
</tbody>
</table>

Table 1: Result of using different approaches to TabletGaze dataset. Retrieved from (Krafka et al, 2016)

1.2 Objectives

Considering the advancement of eye tracking technology and the expanding eye tracking requirements, I’d like to find an efficient and affordable way for people to use eye tracking function on their computers.
Overall, the project is divided into three parts. First, I explored various existing portable hard-based eye trackers or software-based eye trackers, and finished preliminary work such as producing the workflow for using it in computers, to get a baseline eye tracker. Then, I chose a high-quality hard-based eye tracker Tobii T60 as the ground truth, and tried to collect data using the two eye trackers at the same time. At last, I applied some machine learning techniques to improve the baseline eye tracker.
2. Literature Review

2.1 Evaluation of Existing Eye Trackers

2.1.1 Hardware-based Eye Trackers

Many existing eye trackers in the market, priced at between 100 to 10,000 USD, can be classified into two categories, professional eye trackers and portable eye trackers. Professional eye trackers are extremely heavy and inconvenient to use. Tobii T60, for example, with an accuracy of 0.3 degrees, is sold at around 10,000 USD. It has a large system main box and the eye tracking module is also inside its screen, making it inconvenient to use and unaffordable to most people.

![Tobii T60 eye tracker](image)

Figure 3: Tobii T60 eye tracker, which works as the ground truth for this project

With the rapid development of Human-Computer Interaction (HCI) field, an eye tracker with lower accuracy and better portability to fulfill the common demands of different applications is necessary. As a result, various portable eye trackers were released recently. For instance, Eye Tribe, which has an accuracy of 0.5 degrees. Despite the accuracy of Eye Tribe being worse than Tobi T60, it is
compatible with most eye tracker applications and also affordable at only 100 USD, which makes it suitable for most applications (Ooms et al., 2015).

![Eye Tribe](image)

Figure 4: Eye Tribe, with small size, feasible for using with a computer

Considering all the factors discussed above, portable eye trackers should have an edge in the market share from those professional heavy eye trackers. However, portable eye trackers require connection to a USB port for power and data transfer, which limits their practicability to some extent. Overall, a professional eye tracker will be a good choice for research purpose and a portable eye tracker can be used in daily life.

### 2.1.2 Software-based Eye Trackers

As the embedded camera of smartphone and desktop has become more professional in recent years, some developers started to implement eye-tracking softwares that use an embedded camera to make the prediction. Most software-based eye trackers combine face detection and pupil analysis to construct a prediction model by using machine learning algorithms (Meng and Zhao, 2017).
Although more convenient to deploy, existing software-based eye trackers have many drawbacks. Firstly, they cannot provide high frequency with an FPS (frames per second) of 10 to 15 while most hardware-based eye trackers have an FPS of 60. The low FPS makes it hard for software-based eye trackers to handle some contexts that require continuous eye tracking. Secondly, software-based eye trackers cannot produce accurate predictions. For example, WebGazer from Brown University, when its accuracy is 210 pixels in desktop (Papoutsaki et al., 2016), it may not be good for most applications.

2.2 Market Analysis

In the past decade, the demand for eye trackers has been increasing. The global eye tracking market is expected to be worth 1,376.5 Million USD by 2023 (Marketsandmarkets.com, 2017).

A prosperous market may be the result of advancement in HCI (Ghaoui, 2006), which encourages researchers to use eye-tracking technology daily. Actually, there are many useful softwares in the
market that use eye-tracking technology. For example, Sydney Techlab designed a simple program to allow automatic page turning based on Eye Tribe two years ago. This kind of applications would be very helpful in the daily life because it improves user experience to a great extent. Thus it is expected that the future demand for eye tracker mainly would come from HCI field (Majaranta and Bulling, 2014).

Currently, researchers are trying to find suitable eye trackers for HCI applications. Since this kind of applications commonly does not require high accuracy, using professional eye trackers or portable eye trackers may be a waste of resources. Therefore, some researchers start to work on improving the accuracy of the software-based eye trackers. Comparing to hardware-based eye trackers, software-based eye trackers are more user-friendly. A well-developed software-based eye tracker can serve millions of people with a low marginal cost. Based on the factors presented above,

Figure 6: The report about eye tracking market according to technavio.
software-based eye trackers are very likely to dominate the market of eye tracker in the future, on condition that its accuracy would become satisfying in the next few years.

2.3 Problem Scope

2.3.1 Failed Attempts

As the goal of my project is to provide a way for efficient eye tracking in desktop by means of machine learning. Initially, we chose Eye Tribe, which is a portable eye tracker, as the eye tracker to improve. Considering the size and the electricity consumption, Eye Tribe is acceptable.

However, when I performed data collection, both of them cannot output eye gaze normally. After several experiments, I found that the infrared ray pattern of Eye Tribe and Tobii T60 would disturb each other, hence both of them cannot work normally.
Afterwards, I also tried to use other high-quality hardware-based eye trackers as the ground truth to do data collection. Some of them use matrix-shape infrared ray pattern (SMI RedN), some of them use single point infrared ray pattern (EyeLink). However, all of the attempts failed. Finally, we drew a conclusion that it is feasible to synchronize to infrared ray based eye trackers to do data collection. After discussing with Dr. Antoni Chan, we decided to change the eye tracker to improve. Noticing the rapid advancement in CNN, we believed that software-based eye tracker is of better prospect.

2.3.2 Problem Statement & Solution

Although being with a good prospect, software-based eye trackers still have a long way to go. Existing software-based eye trackers do not produce a satisfying result, thus improving the accuracy
of them has been an urgent task for the market. Unlike traditional eye trackers, software-based eye trackers do not rely on optics, the accuracy of them mainly depends on the algorithms. In order to improve the prediction accuracy of software-based eye trackers, researchers would focus on optimizing its algorithms, as Zhu and Ji explained (2005).

Existing software-based eye trackers usually assign weight to the images of face and eyes, then make the prediction based on the pre-defined model. Defining the model is always a hard task, even small difference in pupil can cause eye fixation to move a long distance in the screen, it is not easy for the model to response correctly based on such a small shift in the pupil in a complex context.

In order to make the model more ‘comprehensive’ for handling different situations, iTracker from CSAIL group in MIT will be chosen as the prototype (Krafka et al., 2016). The iTracker was trained by using data from 1471 participants and is regarded as the most solid eye-tracking model up to now. In the data collection experiment of iTracker, experimenters would look at a sequence of dots on the screen. Then Krafka et al. (2016) used the recorded face frames from the camera with the coordinates of dots to train a model. In this way, the eye gaze would exactly lie in the dot. However, the environment in the experiment is too ideal. In reality, the eye gaze rarely stays still when people

![Figure 9: Raw prediction from iTracker for 9-points pattern without](image)

17
are looking at the screen. As a result, the iTracker model would fail to produce an accurate result when the eye gaze moves constantly.

To make iTracker more reliable in daily use, the methodology for data collection will be modified. In my experiment, the participants will look at different images on the screen in a natural way. During this process, the embedded camera will record the frames of the face and a high-quality eye tracker will record the eye gaze of the experimenter. When training the model, the eye gaze data from high-quality eye tracker will be used as the reference for data from iTracker. In this process, some machine learning algorithms will be used to remove the noise of the output data from iTracker.
3. Preliminary Work

3.1 Mapping *iTracker* for Computers

As *iTracker* is designed for mobile phone and trained using images taken by the front-end camera of mobile phone, it needs to be modified so that we can use it in computers.

3.1.1 Preparing Inputs

The original *iTracker* is bound with iOS, since it uses the built-in face and eye detection from iOS. However, those functions are not available at computers, so I have to implement the pipeline myself. Although openCV provides a simple face and eye detect function, it is not accurate. So, I chose the face-landmark-localization package for detecting the face and generating a facial landmark.

![Figure 10: openCV eye detecting function (left), face-landmark-localization package (right)](image)

As shown in the Figure 9, the generated facial landmark is a fixed 68 points shape, so I can locate points that represent the eyes and facial outline, and then crop them. Then I can use those points to crop the images of eyes and faces, and also generate the face grid that represents the position of the face in the whole image.
3.1.2 Calibration Setup

Calibration is a very important step in eye tracking, where the geometric characteristics of a user’s eyes and the position properties of the working environment are estimated, so the eye tracker can produce a fully-customized and accurate gaze points.

There are various kinds of calibration methods, such as ridge regression, linear regression and SVM. According to my experiments, when the number of points is small, linear regression is the best choice. Ridge regression is also an option, but it works fine only when the number of points is large. As for SVM, some researchers are able to produce very good performance using it. However, it is very hard to find the parameters with good generality. Considering all the factors, I use ridge regression and set the number of calibration points as 34, which refers to a 25-points pattern with another 9-points pattern at the end.
My laptop is a 13-inch version MacBook Air, with a resolution of 1440 by 900. So the error of ~100 px after calibration is ~2 cm in the screen, which is relatively good and better than most of existing eye tracking packages for computers.

Although the original *iTracker* for iOS devices can work without calibration, the lack of calibration will do harm to the accuracy. When transformed and deployed in computers, the calibration process become more important and even indispensable. If we observe the predicted result from *iTracker*, we can easily find that the result is not good, as the ground truth is a standard 9-points pattern. As *iTracker* is trained for mobile, a calibration is necessary for using it in computers.
Since most eye trackers integrate the calibration process in their own SDKs, I wrote a program for doing calibration for iTracker. There are three kinds of calibration available, which are 9-points, 25-points and 34-points mode respectively. After launching the program, the user needs to look a sequence of red points on the screen. Afterwards, the captured images will be saved to specified directory for calibration.
4. Methodology

4.1 Data Collection

When performing data collection, Tobii T60 will be used as the high-quality eye tracker and MIT 300 will be used as testing images. During the experiment, the experimenter will look at a series of images with Tobii T60 recording the eye fixation and a mobile phone will record a video of experimenter’s face. I plan to make each image stay on screen for 3 seconds, with a 2-second cross fixation between two adjacent images. As shown in figure 13, the cross fixation will give experimenter some time to relax and ensure experimenter look at the central point before showing a new image. After the data collection, I will use the recorded video to get the predicted coordinates from iTracker, which will be regarded as the noisy sequence.

Figure 13: Cross fixation

One tough task of the data collection process is the synchronization between the Tobii T60 and the iTracker. As the video for iTracker is taken by a mobile phone, it is hard to map the timestamps of noisy data and ground truth.

The most intuitive method to solve this problem is to manually match the two sequence by testing different matching points in a certain time period and choose the one that minimize the testing error.
However, in this way the workload for data collection would be much too heavy. In addition, it is possible that the choice that minimize the testing error is wrong, which would affect the accuracy of training data.

To address the problem, beep sound is chosen as the reference for the synchronization between two data sequences. During the data collection experiment, a Python program that generates periodic beep sound will be executed on Tobii T60. In this way, I can get a timestamp for each beep sound generated by the program. Also, I can locate the timestamp for the beep sound in the video taken by mobile phone, by extracting the audio frequency diagram from the video.

![Figure 14: Experiment settings](image)

As shown in the figure 6, since I put the mobile phone at a distance of around 30 cm from the speaker that generate the beep sound, the transmission delay of the sound (around 1 ms) can be negligible. Thus, I only need to locate the timestamp when the frequency of the beep sound changes to make them synchronized. There may be some delay in this part, but this delay can be observed after doing enough experiments as it should be stable.
4.2 Data Processing

As stated in the previous section, since the data sequences from iTracker (comes from the video) and Tobii are labelled with different timestamps, we need to make them synchronize to each other manually. In order to present a generic way for this problem, we use sound wave to do synchronization. After doing the data collection, frames extracted from the video are processed to generate the inputs for iTracker. At the same time, a wav file is generated from the video. Afterwards, the spectrogram of this wav file will be plotted to help synchronize two eye trackers.

Since the sample rate of the wav file, which is 44100HZ, is constant. After we select the synchronization point, the timestamp of this corresponding point at the video can be calculated using the sample rate and the spectrogram index of this point. As the beep sound is generated by the Tobii machine, we can also get the timestamp of this point in Tobii machine.
4.3 Time-Series Model

The eye gaze sequence contains a sequence of continuous coordinates, so it can be treated as a time-series. And thus, a time-series model can be used to de-noise the sequence. This model should be an unsupervised method that receive the L(t) (Low quality sequence from iTracker) as input and output the filtered sequence. Considering the properties of the data type, we choose Kalman Filter as the time-series model.

4.3.1 Kalman Filter vs Kalman Smoother

Intuitive, the task of estimating high-quality eye gaze at certain time stamp is similar to what Kalman Filter did. Kalman Filter refers to a method for updating the estimate of the state of a system recursively by processing a series of observations. After each observation, a new state estimate can be produced. Actually, Kalman Filter is a variant of linear dynamical system, when the time domain is discretized. It is modeled on a Markov chain, and the noise is modeled as Gaussian noise. The different point between Kalman Filter and HMM is that the latent variable of Kalman Filter is continuous while the latent variable of HMM is discrete.

If we use Kalman Filter here, we need to compute:

\[
\left( H_t \mid L_0 = o_0, \ldots, L_t = o_t \right)
\]

Where H refers to the high-quality eye gaze sequence, L refers to the low-quality eye gaze sequence, and o refers to the observations from iTracker.

Thus, when using Kalman filter, we need to define the window size and then just compute the estimated eye gaze in order. However, if we observe the low-quality eye gaze sequence:
We can find that there are eye blink periods embedded in the eye gaze sequence. However, after an eye blink period, the position of eye gaze may jump for a long distance. So the eye gaze points separated by eye-blink-period cannot be treated as a continuous sequence.

In this situation, if we use Kalman Filter, we need to re-initialize the time-series after each eye blink interval. Thus, for some eye gaze point, there may not be enough measurements to compute it. For example, if there’s only 10 eye gazes between two eye blink period. Then for the first few eye gazes, the information given to compute them may not be sufficient. So we turn to Kalman Smoother.

The different thing between Kalman Filter and Kalman Smoother is that Kalman Filter uses the observations given so far to compute the estimated value of current state, but Kalman Smoother is more like a post-processing method, as it computes the whole sequence of states given all the observations. Since the length of continuous eye gaze in eye tracking is not that long, Kalman Smoother should be a better choice.

### 4.3.2 Expectation-Maximization (EM) Algorithm

EM algorithm is an iterative method for finding the maximum likelihood parameters of a statistical model when the equations cannot be solved directly. In Kalman Filter/Smoother, the EM algorithm is used for parameter estimation based on observations.
\[
x_{t+1} = Ax_t + w_t, \quad w_t \sim W_t = N(0, Q)\\
y_t = Cx_t + v_t, \quad v_t \sim V_t = N(0, R)
\]

Given the two equations above, a Kalman Smoother computes the distributions \(x_0, x_1, \ldots, x_t\) based on the parameters \(A, C, Q, R\), and observations \(y_0, y_1, \ldots, y_t\). EM algorithm can estimate the value of \(A, C, Q, R\) based on observations. Thus, before using Kalman Smoother, we use EM algorithm to do parameter estimation.

### 4.3.3 Fixed-window Time-series Model

In real-time use, we can set two options for the model to follow. The first option is to make the window size a constant. Let \(I\) refers to the index of valid eye gazes since the last eye blink interval and starts from 1. When \(I\) is the multiple of 30, we use the recent 30 observations as the input to launch the EM algorithm, so we can get the parameters for the Kalman Smoother. And then, we use the parameters to smooth these 30 observations to get a smoothed sequence. The structure of this model is shown in Figure 16.

![Figure 16: The structure of Fixed-window Kalman Smoother Model](image)

Intuitively, the fixed-window Kalman Smoother is simple to understand. After doing many experiments, I found that 30 points (one-second) is a reasonable windows size that produces good result. However, when using this model, a group of input to Kalman Filter may not be belong to a
same fixation, which brings two issues. First, as a group of eye gazes may be divided by an eye blink interval, the eye gazes in the two sides of the eye blink interval can be much different, thus may not be continuous. Second, if the eye blink interval is long and the number of available eye gazes before the interval is less than 30, the program will wait until more valid eye gazes come in. In this situation, the latency of the system may be significant. Considering the factors above, we build another more flexible time-series model to make the model able for real-time tracking.

4.3.4 Flexible-window Time-series Model

Considering the eye blink interval, we put forward another model with a flexible window size. In this model, another constraint is that when there’s an eye blink, the program will immediately put the eye gaze sequence to launch a Kalman Smoother.

![Figure 17: The structure of Flexible-window Kalman Smoother Model](image)

When using the flexible-window time-series model, the pro is that it can be used real-time (fixed 1 second latency); the con is that sometimes the program may use some amount of eye gazes to launch a Kalman Smoother, which may bring inaccurate results.
4.4 Regression Model

The time-series model is good for real-time eye tracking. As for off-line eye tracking, since the speed is not a concern anymore, we may use a more complex model. Thus, we built a CNN regression model to further optimize the eye gaze by including saliency map in this model.

4.4.1 Saliency Map

In computer vision area, a saliency map refers to a grayscale that displays the unique quality of every pixel. As shown in Figure 18, if some area has a high grey level or unique color quality in the original color image, then this area will show in the saliency map in an obvious way.

![Image](image.png)

Figure 18: Saliency map and object map

In the regression process from L(t) to H(t), a saliency map generated from the current screenshot may help, because people are always more likely to look at those high saliency area. In terms of producing the saliency map, we use the FASA algorithm from EPFL.
4.4.2 Inputs

To transform $L(t)$ into the input of the regression model, we transform each $L(t)$ into an image. The background of the image is black, and a gaussian blur is applied on the eye gaze position, as shown in Figure 19. Since the experiment is performed on Tobii T60, of whom the screen resolution is 1280*1024. So the resolution of the saliency map and gaze images should also be 1280*1024. Considering the computing power and memory limit of GPU desktop in CS Lab, I downscale the images into 640*512. The inputs of the model will be 4 images, including $L(t)$, $L(t-1)$, $L(t-2)$ and the saliency map, when the saliency map is derived from the current screenshot.

![Figure 19: Images that represents eye gaze](image)

When performing the data collection, the dataset is MIT300 Saliency Benchmark. Since the size of the images do not fit the size of Tobii screen exactly, some part of the screen are left black. So when producing the saliency map, I filled those area with black.
4.4.3 Model Structure

Basically, the input will be three eye gaze images and a saliency map and the output will be a coordinate of predicted eye gaze. Actually it is not a complex task because the information contained in the input and output are small, so we did not use a deep structure here.

Before training, all of the train samples are normalized to make $H(t)$ be (0,0). As all the 4 channels of inputs have the same shape, I concatenate them together to get an image-shape input with 4 channels and have them share the same weight. Thus, the model can explore the internal relationship between them to give a robust prediction based on the limited training data.

Figure 20: The saliency map of a screenshot, the two sides are filled with black
Figure 21: The structure of the regression model
5. Result Analysis

Up to now, I have 5 datasets of collected and length of each one is around 330 seconds. For the time-series model, all the 5 datasets can be used to evaluated the results. As for the regression model, I use half for training and half for evaluation.

5.1 Results of Time-series Model

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Original (px)</th>
<th>Fixed-window model (px)</th>
<th>Improvement (px)</th>
<th>Flexible-window model (px)</th>
<th>Improvement (px)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20180206002</td>
<td>129.790</td>
<td>126.907</td>
<td>2.884</td>
<td>127.685</td>
<td>2.105</td>
</tr>
<tr>
<td>20180206003</td>
<td>139.842</td>
<td>135.543</td>
<td>4.299</td>
<td>136.983</td>
<td>2.859</td>
</tr>
<tr>
<td>20180206004</td>
<td>115.378</td>
<td>111.759</td>
<td>3.619</td>
<td>111.451</td>
<td>3.927</td>
</tr>
<tr>
<td>20180206005</td>
<td>165.892</td>
<td>161.276</td>
<td>4.616</td>
<td>163.195</td>
<td>2.698</td>
</tr>
<tr>
<td>20180206006</td>
<td>122.916</td>
<td>118.610</td>
<td>4.306</td>
<td>119.722</td>
<td>3.195</td>
</tr>
</tbody>
</table>

Based on the results, we can found that the overall performance of fixed-window model is better. When do offline eye tracking or the latency is not strictly constrained, the fixed-window model will be a good choice. In terms of real time eye tracking, flexible-window model is useful and stable for improving the performance.

5.2 Results of Regression Model

<table>
<thead>
<tr>
<th>Training Dataset</th>
<th>Testing Dataset</th>
<th>Original (px)</th>
<th>Regression Model (px)</th>
<th>Improvement (px)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20180206002+20180206003+20180206004</td>
<td>20180206005+20180206006</td>
<td>144.819</td>
<td>131.832</td>
<td>12.987</td>
</tr>
<tr>
<td>20180206004+20180206005+20180206006+20180206003</td>
<td>20180206002+20180206003</td>
<td>134.823</td>
<td>123.023</td>
<td>11.800</td>
</tr>
</tbody>
</table>

The regression model has a good performance. However, since preparing the inputs take a lot of time, it can only be used for offline eye tracking.
6. Future Improvement

6.1 Retrain iTracker

The iTracker is indeed trained from a tremendous dataset. However, it only uses the raw color images as input for the model, and extracts features from them. I believe that it will make sense to add some computer vision tricks to improve it. For example, Histogram of oriented gradient (HOG) feature may be used on the input images to reveal shape of eyes.

In addition, although after porting to computers, the model performs good, it is not trained for computers. The eye gaze patterns for computer and mobile phone are not the same, so it shall be very useful to retrain the model using dataset for computers.

Figure 22: HOG example, can extract the shape of eyes

In addition, although after porting to computers, the model performs good, it is not trained for computers. The eye gaze patterns for computer and mobile phone are not the same, so it shall be very useful to retrain the model using dataset for computers.
6.2 Use Fixation as Input

An eye fixation is constituted by a series of eye gazes that are close in time and space, referring to a period when the eyes are focused on one object. Up to now, this project focuses on raw gaze. However, in real life, an eye fixation is actually more meaningful than an eye gaze, as only a series of continuous eye gazes cannot reveal meaningful information. So it may help to transform the eye gazes into eye fixations, and use the eye fixation as input to perform higher level analysis, so as to extract more useful informations.
7. References


## 8. Appendix

### A. Monthly Logs

<table>
<thead>
<tr>
<th>Month</th>
<th>Task</th>
</tr>
</thead>
</table>
| **September** | 1. Trying synchronize Eye Tribe with Tobii  
                  2. Trying synchronize Eye Tribe with RedN |
| **October**  | 1. Testing the feasibility of using EyeLink eye tracker  
                  2. Investigating the iTracker model  
                  3. Finding model for eye detection and modify the recognition 
                                          algorithm, then prepare the input data for iTracker  
                  4. Trying to run the iTracker model |
| **November** | 1. Using different calibration patterns for iTracker  
                  2. Getting initial results  
                  3. Trying to synchronize the data from Tobii and Video. The 
                                          data are not from same desktop, thus the time stamp is not 
                                          synchronized. |
| **December** | 1. Data collection using mobile phone, with a beep sound.  
                  2. Self-learning on related time series model |
| **January**  | 1. By using beep sound to as the stimulation, synchronize Tobii 
                  and iTracker to get data.  
                  2. Extracting the audio frequency spectrum from the video.  
                  3. Writing program to generate calibration process in Tobii and 
                                          redo data collection. |
| **February** | 1. Previously collected data doesn't include the calibration part of 
                  iTracker, redo data collection for the 300 images with calibration 
                                          for it.  
                  2. The calibration result of iTracker is not good on the Tobii 
                                          screen, currently I am looking for the cause, maybe need to redo 
                                          data collection, can get the result before meeting.  
                  3. Successfully synchronize the data using audio frequency 
                                          spectrum |
<table>
<thead>
<tr>
<th>Month</th>
<th>Task</th>
</tr>
</thead>
</table>
| **March** | 1. Build the time-series model using Kalman Smoother with EM algorithm  
2. Write program for generating the inputs for CNN regression model  
3. Building the CNN regression model  
4. Tune the parameters for the models |
| **April** | 1. Keep refining the model  
2. Write final report |