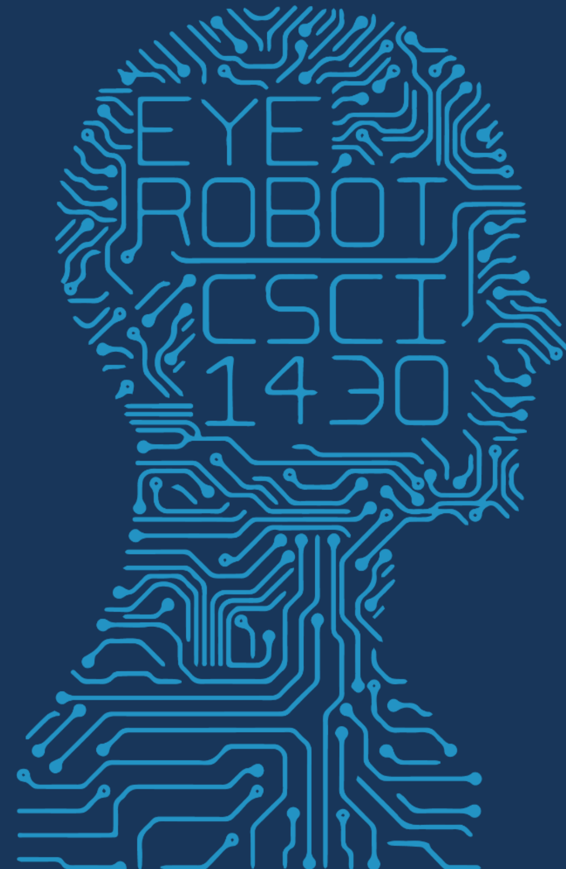

1950

Future Vision




2017 MWF 1pm

Computer Vision

# Course so far

Image formation and color!

Filtering!

Image frequency!

Feature points!

Bags of words!

Classifiers!

Sliding windows!

Big data!

# Course coming up

Neural Nets

Convolutional Neural Nets

- Project 4

Current state of the art

\- - -

Camera geometry

Stereo

- Project 5 (not very long)

# Project 6 - WebGazer

- Team project – of 4 -> no single person teams
  - Show to class on Dec 11th
  - Report/code due Dec 12th
- Starts after project 4 CNNs (~Nov 10th)
- But _organize now_

# WebGazer - https://webgazer.cs.brown.edu/

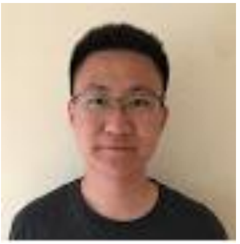
Fork me on GitHub

Pure Javascript real-time eye tracking
Exploits gaze/mouse click interaction coherence

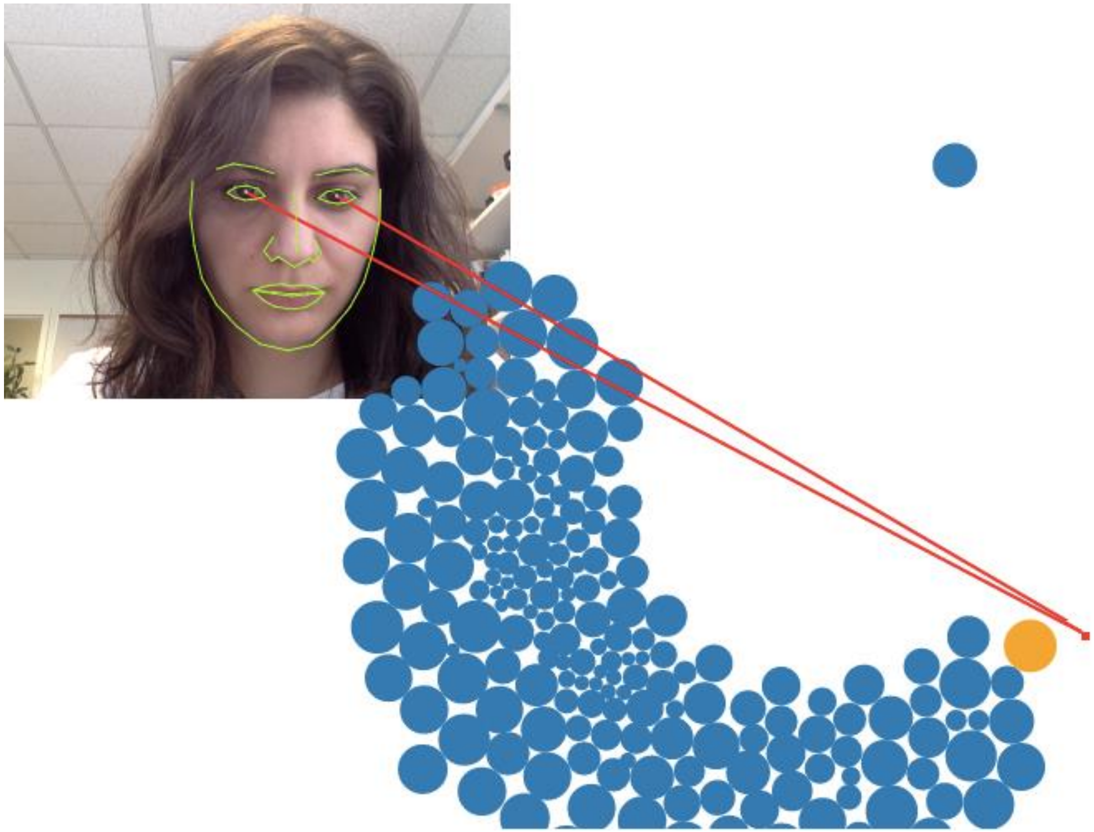
Alexandra Papoutsaki

Jeff Huang

Aaron Gokaslan



Yuze (Harry) He



# Why eye tracking?

Eye gaze is important cue in human-human communication.

-> Implicitly a fundamental technique to future natural computing interfaces

# Some state of the art stuff

Mturk-based CNN for eye tracking

- https://blogs.nvidia.com/blog/2016/08/30/eye-tracking-deep-learning/

AI-based Co-Pilot for driving

- https://www.youtube.com/watch?v=h9npvMFI-mc

Eyetracking for avatar eye capture (e.g., for virtual reality)

Eyetracking for foveated rendering for virtual reality

https://venturebeat.com/2017/09/06/eye-tracking-is-virtual-realitys-next-frontier/

# Some state of the art stuff

Alexandra projects (user behavior analysis):

- Eye tracking for remote studies of Web search

- Eye tracking as a typing aid; for touch typist identification

- Eye tracking as a human development aid,
  as a cue to learning disability or disease

# How does WebGazer work right now?

## Step 1: Detailed face detection

*clmtrackr* -> Javascript learning-based facial feature tracker

Returns image locations of these landmarks.

# How does WebGazer work right now?

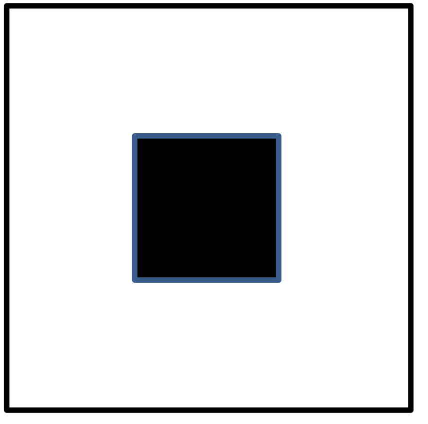
Step 2: Pupil detection

-> Compute *integral image* of eye region

-> Sliding window detector

-> 2D Haar-like feature

Maximize ratio of
inner to outer regions.



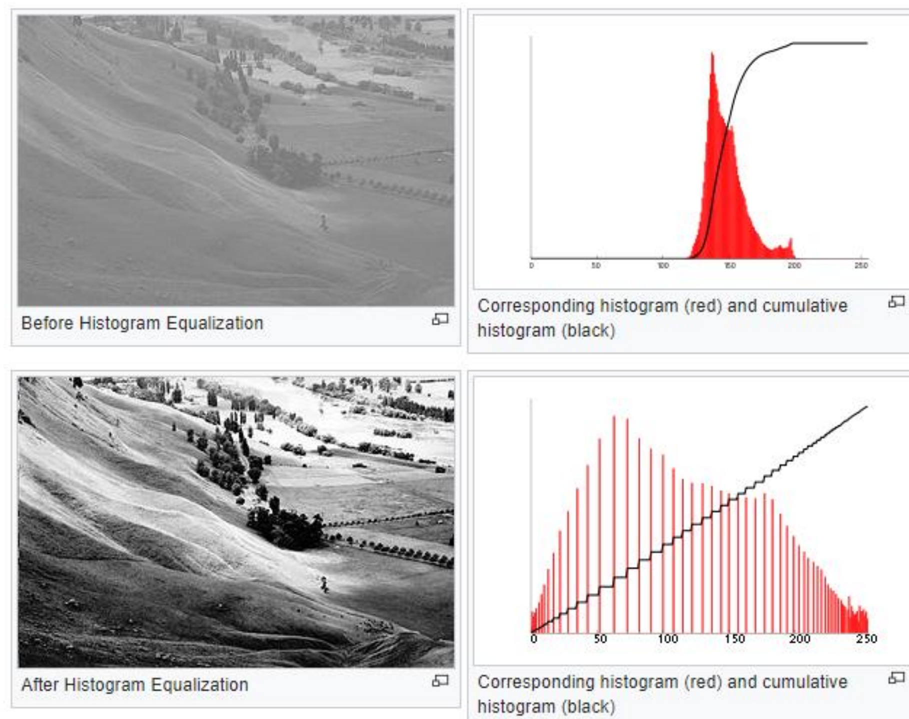
# How does WebGazer work right now?

Step 3: Eye feature (120 dim)

-> Extract 6x10 pixel rectangle around pupil (!)

-> Grayscale intensity

-> Histogram equalization




Before Histogram Equalization

Corresponding histogram (red) and cumulative histogram (black)

After Histogram Equalization

Corresponding histogram (red) and cumulative histogram (black)

# How does WebGazer work right now?

Step 4: Linear regression (with regularization)

Goal: Learn a function which maps
eye feature to screen position.

$f(x) = y$
$x$ = eye feature
$y$ = mouse click data – *you look where you click!*

# Reminder: linear regression

- Eye features $\mathbf{x} = (x_1, ..., x_N)$
- Display click horizontal $\mathbf{t} = (D_{x1}, ..., D_{xN})$

- Estimate $f(\mathbf{v}) = \phi(\mathbf{x})^T \mathbf{w}$

  Regularization!

  s.t. $\underset{\mathbf{w}}{\text{minimize}} \sum_{x_i \in \mathbf{x}} ||D_{xi} - f(x_i)||_2^2 + \lambda ||\mathbf{w}||_2^2$

- Closed-form solution $\mathbf{w} = (X^T X + \lambda I)^{-1} X^T Y$

  (matrix notation)

*Train one function for horizontal, one for vertical.*

# Hypothetical program loop pseudocode

*Thread 1:*

```
while(true)
    eyeloc = clmtracker.trackFace( webcam.getImage() );
```

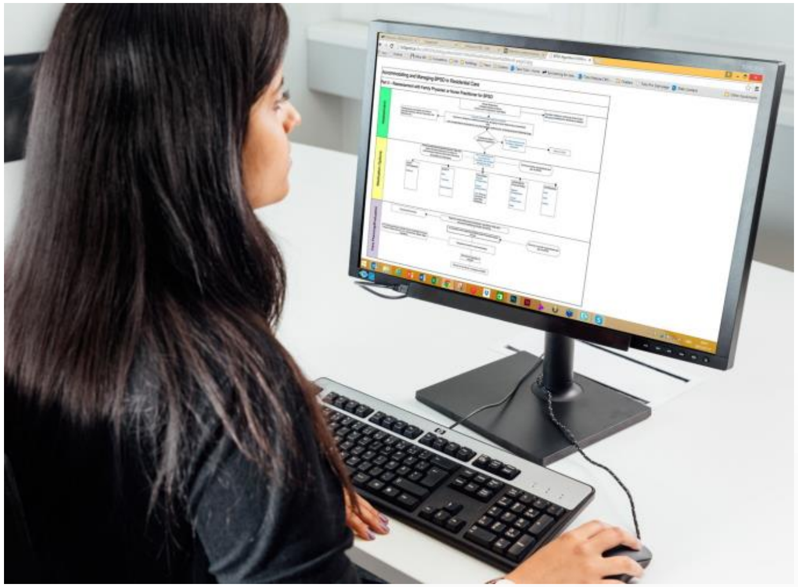
*Thread 2:*

```
allEyeFeats = [];          % Eye feature storage
allClickLocations = [];  % 2D click locations
onMouseClick( MouseEvent me )
    allEyeFeats(i) = extractEyeFeat( findPupil(eyeloc) );
    allClickLocations(i) = me.xy;
    f = linearRegression( allEyeFeats, allClickLocations );
```

*Thread 3:*

```
gaze = predict( f, extractEyeFeat( findPupil(eyeloc) ) );
```

# How do we know if it works?



Tobii Pro X3-120 eye tracker
Accurate to 1 degree at desktop range
~ 1.7 cm
Or ~ 50 pixels at 72 dpi


WebGazer error against Tobii EyeX number is
150 pixel mean, 140 std.dev.

# Can we do better?

- WebGazer assumes *no* prior knowledge
- It learns as you click – advantages/disadvantages?

- Could we improve it in this scenario?
- What about with a little data?

# Training data

51 participants, 30 minutes each @ 30 Hz

Webcam videos

Mouse click data

Tobii Pro X3-120 eyetracking data

Screen captures

Alexandra collected all of this, and wants us to exploit it!

http://cs.brown.edu/courses/csci1430/proj_webgazer/webgazer_data.pdf

# Training data, but show them to me

- Mention the calibration process, James!

# Train / test split

We will give you *some* of the data.

We will use the rest as a testing set to measure both WebGazer's performance and your performance.

# Compute

We will get you some compute.

Still sorting things out…

# Project 6 - WebGazer

- ‘Pure’ challenge
  - Must work in ~real time in browser
  - Must be deployable as Javascript library

- Fallback ‘wild’ challenge
  - No restrictions.

# Project 6 - WebGazer

- It is a real research problem.
- It is multifaceted, and it can be as much of a challenge as you wish.
- You can use anything and everything.

# Jeff’s Carrot

If you can “visibly improve the eye tracking”, and keep the Web/real-time constraints…

…then Jeff has money for you to integrate your work with WebGazer,
for you to become co-authors on the project,
and for you to share the IP.

Go Jeff.

# Rest of today: Challenge discussion

- Medium groups – 6-10 *(not your project groups!)*

- Identify possible WebGazer problems.
- Discuss different solutions.
- Investigate what might be done.
- ‘Back of envelope’ computation costs.
- Write! Sketch! Ask me questions!

- Last 10 minutes: class discussion on what you came up with.

http://cs.brown.edu/courses/csci1430/proj_webgazer/webgazer_data.pdf

# First steps

- Try out WebGazer
- Use the library on a page of your own
- Read the Webgazer paper

http://cs.brown.edu/people/alexpap/papers/ijcai2016webgazer.pdf
Don’t get hung up on things you might not understand yet; barrel through.

- Fork it.
- Test it on the challenge data (next few weeks).