

Tracking Library for the Web

Eduardo A. Lundgren Melo



UNIVERSIDADE
FEDERAL
DE PERNAMBUCO



Master of Science in Computer Science

Silvio de Barros Melo (*Advisor*)
Veronica Teichrieb (*Co-Advisor*)



Outline

1 Introduction

2 Basic concepts

- Web
- Visual tracking
- Visual tracking on the web

3 Tracking library for the web

4 Results and evaluation

- Color tracking algorithm
- Rapid object detection (Viola Jones)
- Markerless tracking algorithm

5 Conclusions

Outline

1 Introduction

2 Basic concepts

- Web
- Visual tracking
- Visual tracking on the web

3 Tracking library for the web

4 Results and evaluation

- Color tracking algorithm
- Rapid object detection (Viola Jones)
- Markerless tracking algorithm

5 Conclusions

Motivation

- The web browser environment is evolving fast
- Phones and notebooks devices have embedded web browser
- Entertainment solutions are gaining space on the web
- Vision is an accurate and low-cost solution

Problem definition

- Visual tracking requires video capturing and processing
- Video processing requires high computational complexity
- JavaScript is a language interpreted by all web browsers
- Interpreted languages have limited computational power

Objectives

- Facilitate user interaction with the web browser
- Accelerate the use of visual tracking in commercial products
- Implement a cross-platform tracking library for the web



Outline

1 Introduction

2 Basic concepts

- Web
- Visual tracking
- Visual tracking on the web

3 Tracking library for the web

4 Results and evaluation

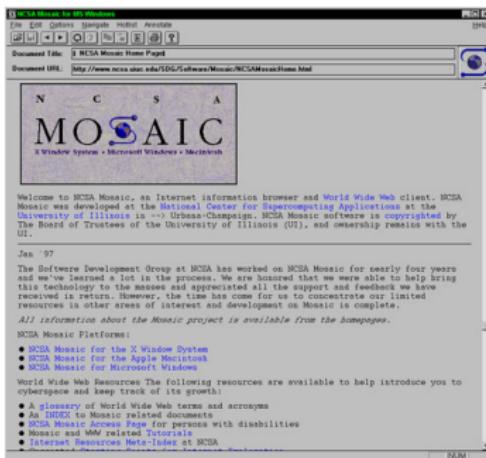
- Color tracking algorithm
- Rapid object detection (Viola Jones)
- Markerless tracking algorithm

5 Conclusions



Web

The beginning of the web



The beginning of the web

- Plain text and images were the most advanced features
 - In 1994, the World Wide Web Consortium (W3C) was founded
 - Companies were able to contribute to the W3C specifications
 - Today's web is a result of the ongoing efforts of an open web

The modern web





Browser technologies





Visual tracking

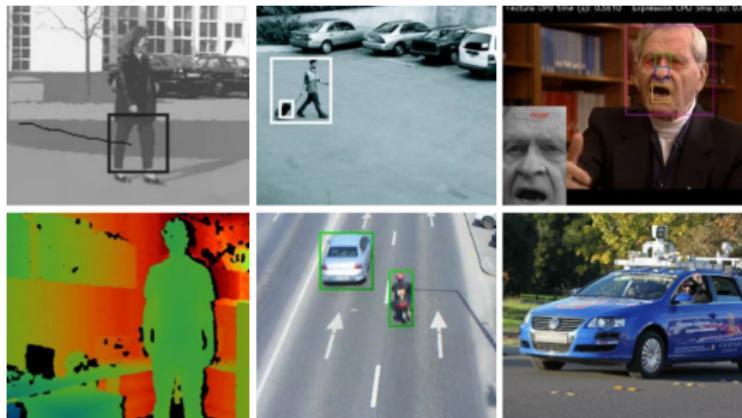
Visual tracking





Visual tracking

Visual tracking

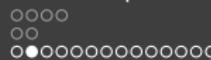




Visual tracking on the web

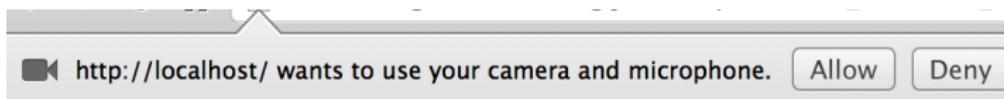
Visual tracking workflow on the web



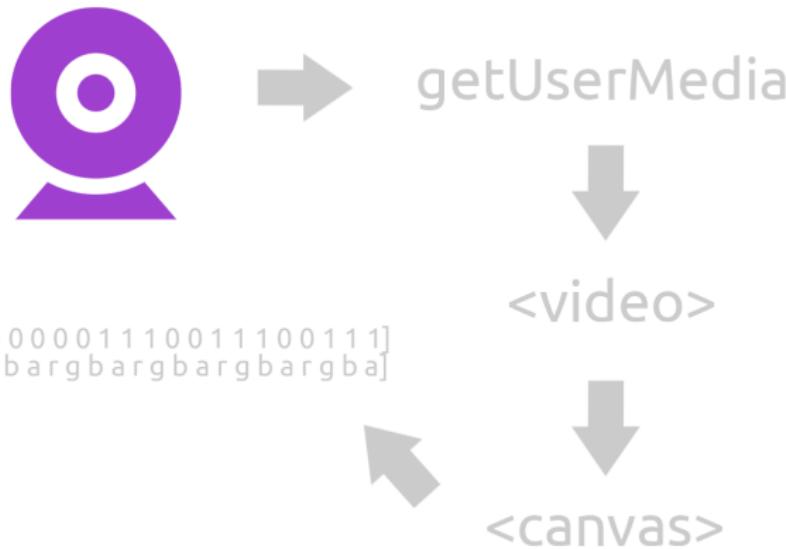


Visual tracking on the web

1. Request user web-cam access



1. Request user web-cam access



2. Capture web-cam stream





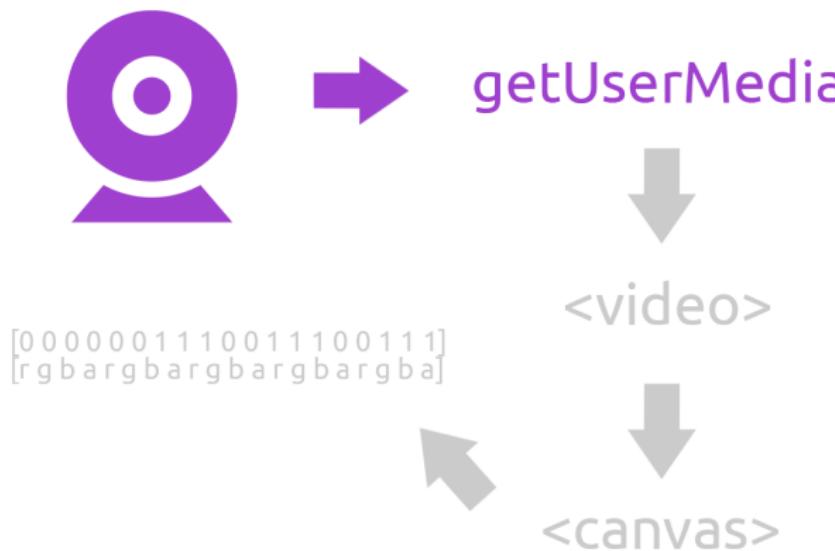
2. Capture web-cam stream

```
1
2   <script>
3     navigator.getUserMedia({ video: true }, function(localMediaStream) {
4       // Stream captured
5     }, onFail);
6   </script>
```



Visual tracking on the web

2. Capture web-cam stream





Visual tracking on the web

3. Reproduce web-cam stream into the video





3. Reproduce web-cam stream into the video

```
1 <video autoplay></video>
2 <script>
3   var video = document.querySelector('video');
4   navigator.getUserMedia({video: true}, function(localMediaStream) {
5     video.src = window.URL.createObjectURL(localMediaStream);
6     video.onloadedmetadata = function(e) { alert('Ready to go.') };
7   }, onFail);
8 </script>
```



3. Reproduce web-cam stream into the video

```
1 <video autoplay></video>
2 <script>
3     var video = document.querySelector('video');
4     navigator.getUserMedia({video: true}, function(localMediaStream) {
5         video.src = window.URL.createObjectURL(localMediaStream);
6         video.onloadedmetadata = function(e) { alert('Ready to go.') };
7     }, onFail);
8 </script>
```



Visual tracking on the web

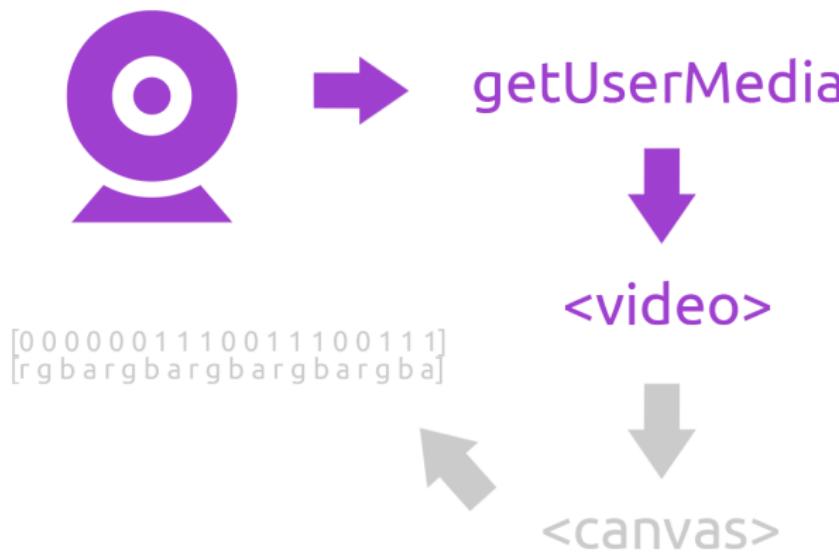
3. Reproduce web-cam stream into the video

```
1 <video autoplay></video>
2 <script>
3     var video = document.querySelector('video');
4     navigator.getUserMedia({video: true}, function(localMediaStream) {
5         video.src = window.URL.createObjectURL(localMediaStream);
6         video.onloadedmetadata = function(e) { alert('Ready to go.') };
7     }, onFail);
8 </script>
```



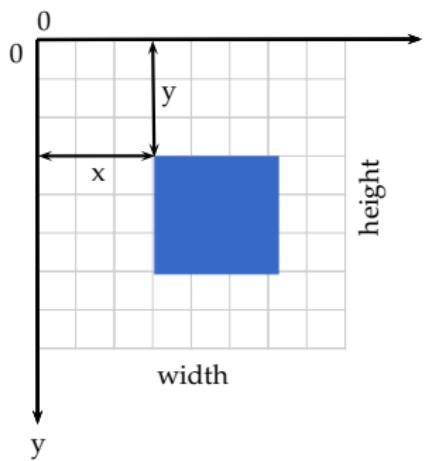
Visual tracking on the web

3. Reproduce web-cam stream into the video





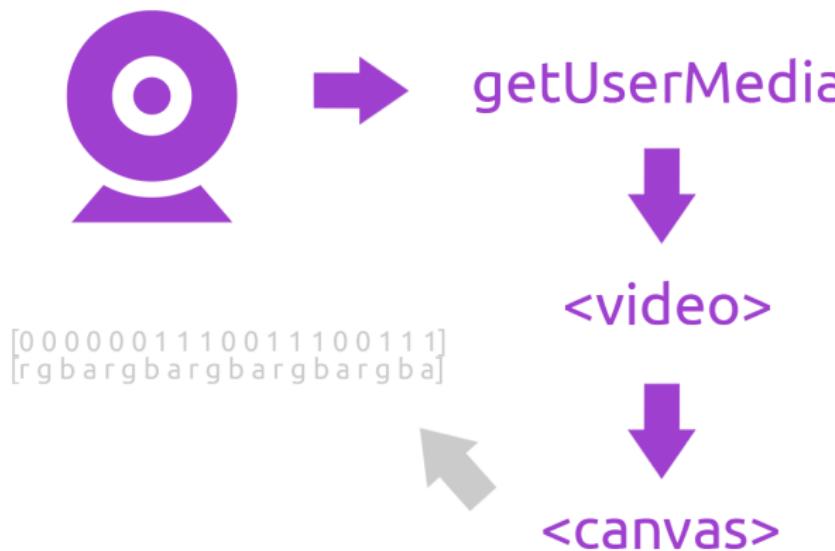
4. Process video data using canvas





Visual tracking on the web

4. Process video data using canvas





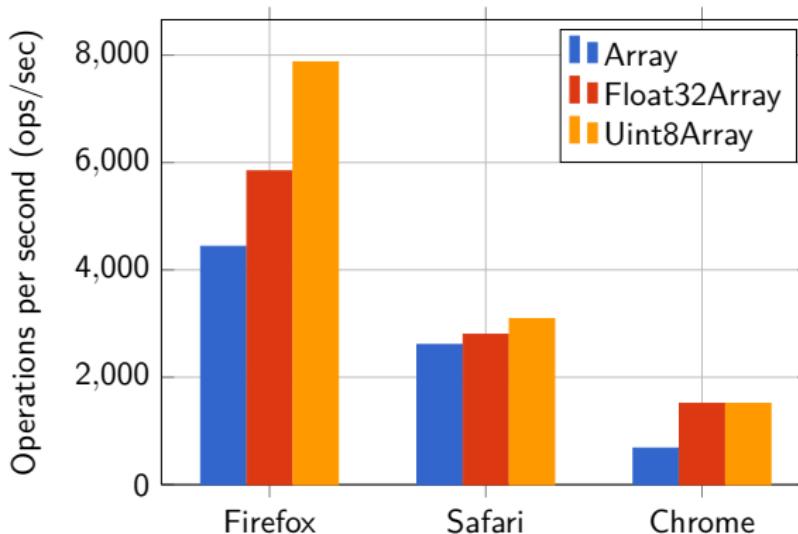
5. Access canvas data using JavaScript typed arrays

- In the past, raw data was accessed as a string
- Browsers needed a quick way to manipulate raw binary data
- Typed data structures were added to JavaScript
- JavaScript-typed arrays access raw binary more efficiently





5. Access canvas data using JavaScript typed arrays





Visual tracking on the web

5. Access canvas data using JavaScript typed arrays



getUserMedia



<video>

[0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 1]
[r g b a r g b a r g b a r g b a]



<canvas>



Outline

1 Introduction

2 Basic concepts

- Web
- Visual tracking
- Visual tracking on the web

3 Tracking library for the web

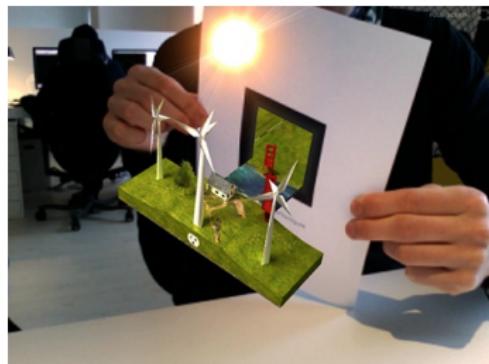
4 Results and evaluation

- Color tracking algorithm
- Rapid object detection (Viola Jones)
- Markerless tracking algorithm

5 Conclusions

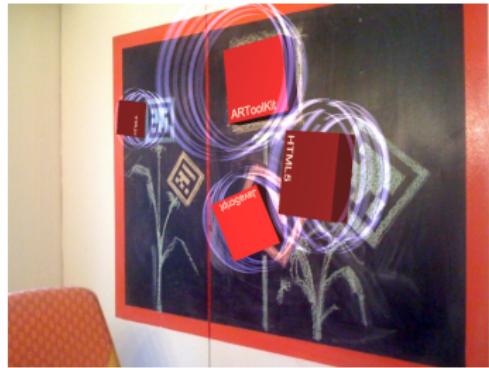
Related work

- FLARToolKit: a port of ARToolKit marker tracking library to ActionScript



Related work

- JSARToolkit: is a JavaScript port of FLARToolKit

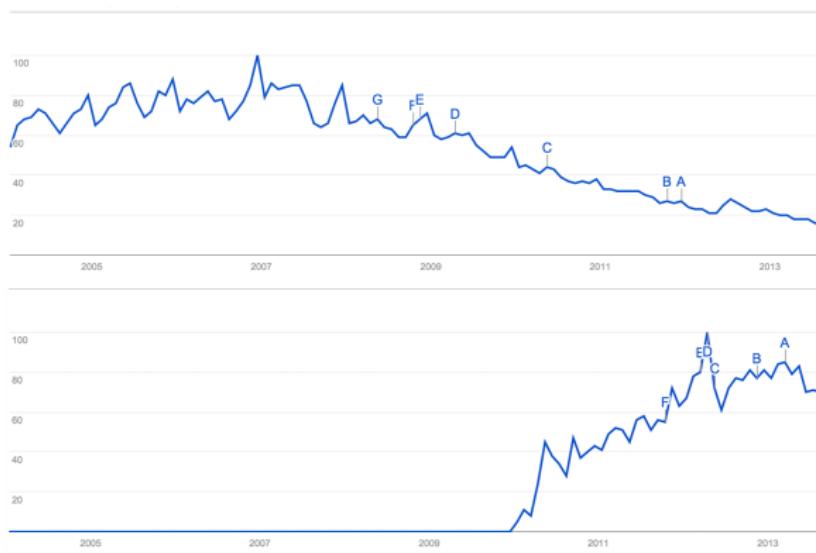


Related work

- Unifeye Viewer: a robust markerless tracking solution for the web to ActionScript



Flash vs HTML5



○○○
○○
○○○○○○○○○○○○○○

tracking.js

tracking.js About Examples Download now Fork on Github

tracking.js

Change the way you interact with your browser



Download now Fork on Github

Example

Basic Usage

This example is a simple way to initialize the user browser camera and start tracking for objects with color **magenta**. There are two available callbacks, **onfound** is fired when the object is detected and **onlosttrack** does the opposite. The **onframe** callback receives as argument a track.

```
var videoCamera = new tracking.VideoCamera().render().renderVideoCanvas();  
  
videoCamera.track({  
  type: 'color',  
  color: 'magenta',  
  onfound: function(track) {  
    console.log('track.x, track.y, track.z');  
  },  
  onlosttrack: function() {}  
});  
  
onframe: function() {
```

tracking.js

Tracking library for the web

Common infrastructure to develop visual tracking applications and to accelerate the use of those techniques on the web in commercial products.

Library features

1. Color tracking

Library features

1. Color tracking

2. Rapid object detection (Viola Jones)

Library features

1. Color tracking

2. Rapid object detection (Viola Jones)

3. Markerless tracking algorithm

Color tracking algorithm



Figure : © <http://www.flickr.com/photos/laynecom/8674644879/>

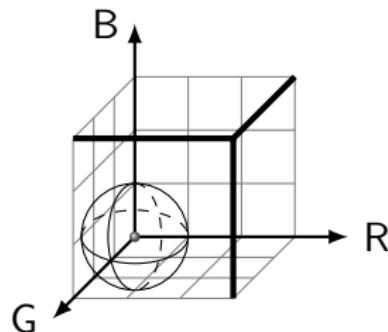
Color tracking algorithm

```
1  var videoCamera = new tracking.VideoCamera();
2  videoCamera.track({
3      type: 'color',
4      color: 'magenta',
5      onFound: function(track) {
6          // do your logic here.
7      }
8  });
```



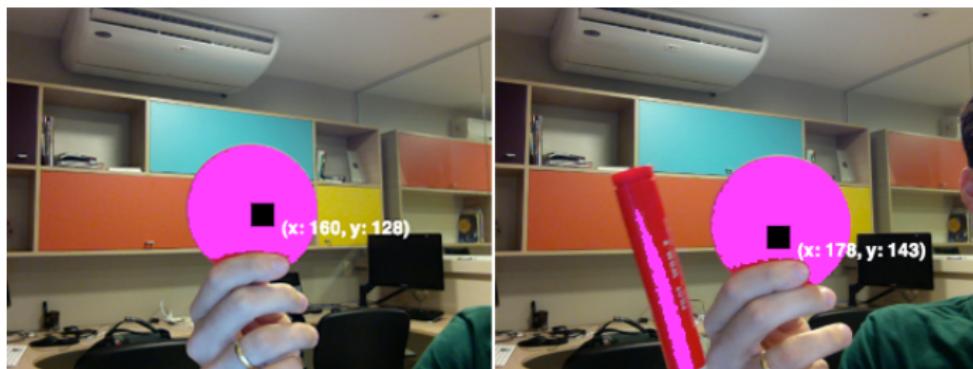
Color tracking algorithm - Color difference evaluation

$$\|C_1 - C_2\| = \sqrt{(C_{1,R} - C_{2,R})^2 + (C_{1,G} - C_{2,G})^2 + (C_{1,B} - C_{2,B})^2}$$

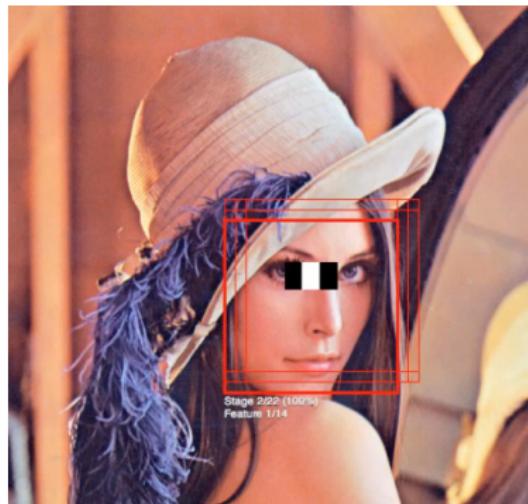




Color tracking algorithm - Color blob detection



Rapid object detection (Viola Jones)

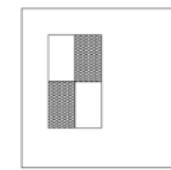
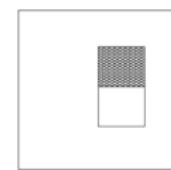
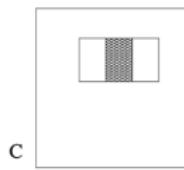
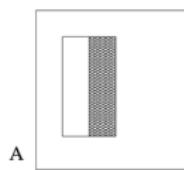
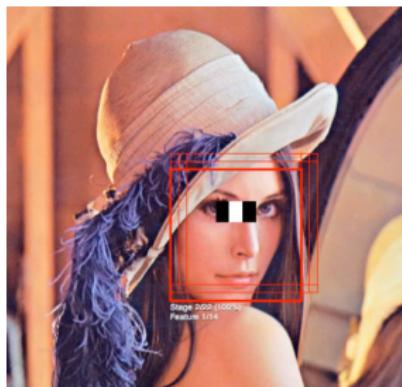


Rapid object detection (Viola Jones)

```
1 var videoCamera = new tracking.VideoCamera();
2 videoCamera.track({
3     type: 'human',
4     data: 'frontal_face',
5     onFound: function(track) {
6         // do your logic here.
7     }
8});
```



Rapid object detection (Viola Jones)



Rapid object detection (Viola Jones)

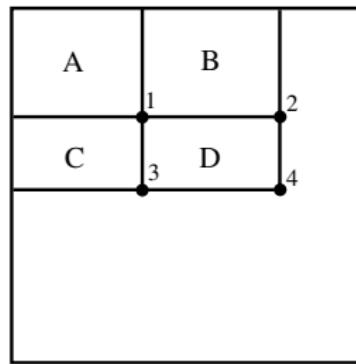
Integral Image

Rectangle features can be computed very rapidly using an intermediate representation for the image which we call the integral image.

The integral image at location x, y contains the sum of the pixels above and to the left of x, y , inclusive

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

Rapid object detection (Viola Jones)



Rapid object detection (Viola Jones)

Scanning detector algorithm

- 1 Create or scale a 20×20 squared block by 1.25 per iteration
- 2 Loop the block by Δ pixels over the image
- 3 For each block location, loop the tree and evaluate each stage
- 4 Positive stage evaluate next stage, otherwise stops the loop
- 5 If all stages were positive store the rectangle
- 6 Once the tree is done, group the overlapping rectangles
- 7 Find the best rectangle of each the group (merging phase)

Rapid object detection (Viola Jones)

1. Optimization: merging phase

Rectangles are used partitioned into a disjoint set data structure.
On this work it was replaced by an alternative called Minimum Neighbor Area Grouping.

Rapid object detection (Viola Jones)

1. Optimization: merging phase

Rectangles are used partitioned into a disjoint set data structure. On this work it was replaced by an alternative called Minimum Neighbor Area Grouping.

2. Optimization: training data

OpenCV uses XML to store training data. This work proposes the usage of row-major order typed arrays.

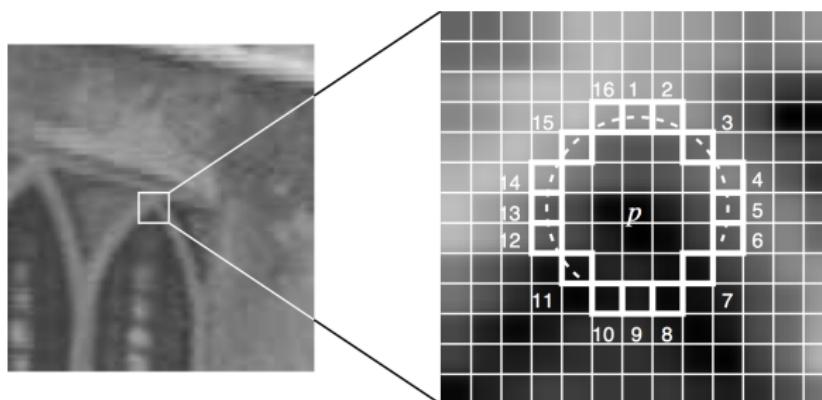
```
oooo
oo
ooooooooooooooo
```

Rapid object detection (Viola Jones)

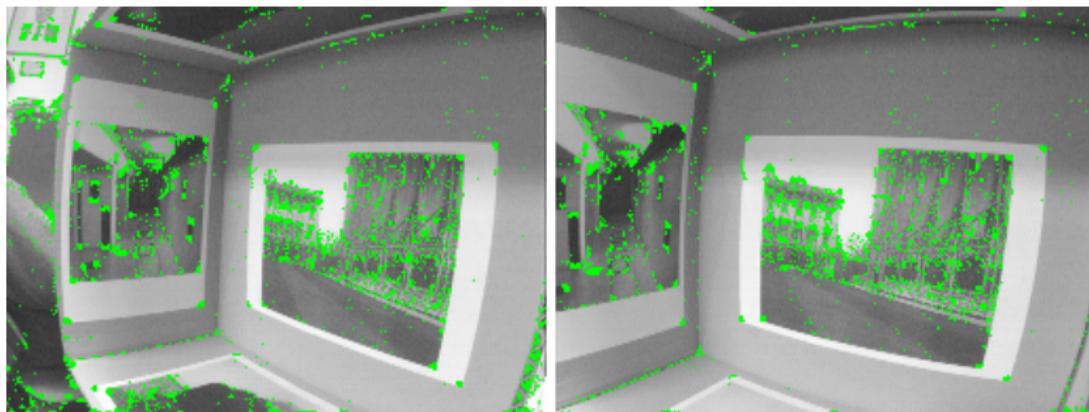
```
haarcascade_frontalface_alt.xml
 46 <haarcascade_frontalface_alt type_id="opencv-haar-classifier">
 47   <size>20</size>
 48   <stages>
 49     <!-- stage 0 -->
 50     <trees>
 51       <!-- tree 0 -->
 52       <!-- tree 1 -->
 53       <!-- tree 2 -->
 54       <!-- tree 3 -->
 55       <!-- root node -->
 56     <features>
 57       <rects>
 58         <>3 7 14 4 -1,</>
 59         <>3 9 14 2 2,</>
 60       </rects>
 61       <tilted=></tilted>
 62     </threshold>4.0141958743333817e-003</threshold>
 63     <left_val>0.833794197346249</left_val>
 64     <right_val>0.8378106951713562</right_val></>
 65   <!-- tree 1 -->
 66   <!-- tree 2 -->
 67   <!-- root node -->
```

```
frontal_face.js
 1 [ [ -1, 0.8378106951713562, [ [ 7, 14, 4, -3, 0, 14, 2, 8, 0.8049119587433338, 0.8378106951713562, 0.15123098380989, 0.151423228357669, 0.7488012208175659 ], [ 1, 7, 15, 9, -1, 1, 18, 15, 3, 3, 0.004218993103194891, 0.69049428177363467, 0.6734819874763408 ] ] ], [ 0, 6, 956668772277832, [ [ 6, 2, 6, -1, 5, 9, 2, 1, 2, 0, 0016227109509545297, 0.06939885864106287, 0.71109461784362791 ], [ 7, 5, 6, 3, -1, 9, 5, 2, 3, 3, 0, 00220966439328919, 0.1795830318463715, 0.6668662231178284 ], [ 4, 0, 12, 9, -1, 4, 3, 12, 3, 3, 0, 0050825760842680517, 0.169367298483848, 0.655406934165959 ], [ 6, 9, 18, 8, -1, 6, 13, 10, 4, 2, 0, .007965989418677228, 0.583332054138184, 0.09141451865434651 ], [ 3, 6, 14, 8, -1, 3, 10, 14, 4, 2, 0, -0.03522701805789671, 0.143166079014572, 0.6831095879393978 ], [ 14, 1, 6, 10, -1, 14, 1, 5, 10, 2, 0.0001361747381221, 0.6101385774612427, 0.2088589547061530 ], [ 1, 1, 18, 3, -1, 13, 6, 9, 3, 0, 0.008696131400214569, 0.2836230932780874, 0.43640273957252502 ], [ 1, 8, 17, 2, -1, 1, 9, 17, 1, 2, 0, 0.0011488880263641477, 0.222358026655723, 0.58007007837329553 ], [ 16, 6, 4, 2, -1, 16, 7, 4, 1, 2, 0, 00212196862986483, 0.5596548131862892, 0.116223703622818 ], [ 14, 2, 6, 12, -1, 14, 2, 3, 12, 2, -0.0939416565725, 0.6580273728370665, 0.4717746110583711 ], [ 4, 0, 4, 12, -1, 4, 0, 2, 6, 2, 0, 6, 2, 6, 2, 0.00137778942696758, 0.5993673801422119, 0.834529876708084 ], [ 2, 11, 18, 6, 1, 8, 11, 6, 8, 8, 0.730631574988365, 0.434188643545834, 0.7060834273854932 ], [ 5, 7, 18, 2, 1, 5, 15, 12, 1, 3, -0.0860011800963545, 0.17084139594841, 0.567525684833268 ], [ 1, 1, 9, 0.40854278564451, [ [ 3, 10, 9, 4, -1, 6, 10, 5, 1, 0, 0.165106806311045, 0.6644225170544434, 0, 1424857974052429 ], [ 9, 4, 2, 14, -1, 9, 11, 2, 7, 3, 0.0270524993588565, 0.6325352191925049, 0,
```

Feature detector (FAST)



Feature detector (FAST)



Feature detector (FAST)

1. Optimization: avoid machine learning

The steps required for machine learning involves extra math operations, such as *log*. The results were good without it.

Feature detector (FAST)

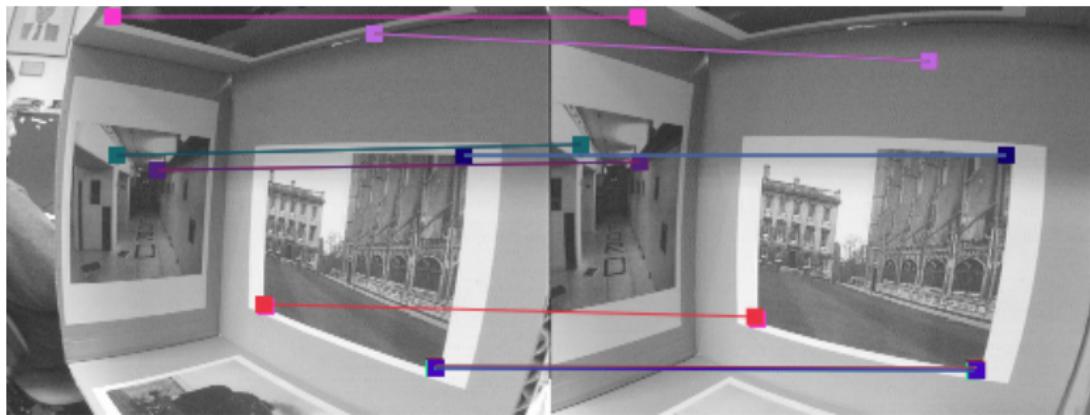
1. Optimization: avoid machine learning

The steps required for machine learning involves extra math operations, such as *log*. The results were good without it.

2. Optimization: avoid non-maximal suppression

To remove corners which have an adjacent corner requires extra access to the typed array.

Feature extractor (BRIEF)



Feature extractor (BRIEF)

To generate the binary strings it is defined the test τ on patch \mathbf{p} of size $\mathbf{S} \times \mathbf{S}$ as:

$$\tau(\mathbf{p}; x, y) := \begin{cases} 1 & \text{if } \mathbf{p}(x) < \mathbf{p}(y), \\ 0 & \text{otherwise} \end{cases}$$

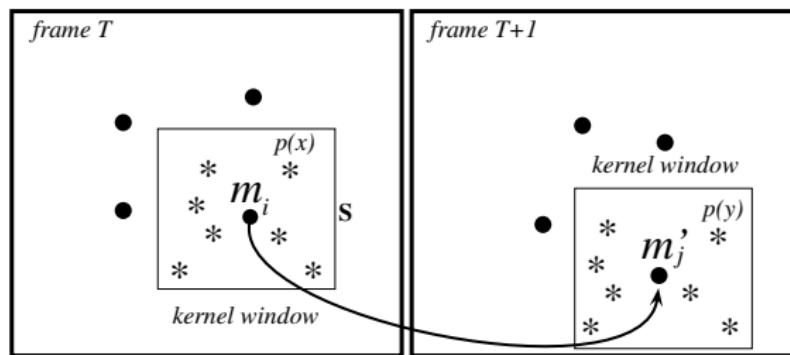
Feature extractor (BRIEF)

The n_d -dimensional bit-string is our BRIEF descriptor for each keypoint:

$$f_{n_d}(\mathbf{p}) := \sum_{1 \leq i \leq n_d} 2^{i-1} \tau(\mathbf{p}; x, y).$$

In this work $n_d = 128$ was used. The number of bytes required to store the descriptor can be calculated by $k = n_d/8$.

Feature extractor (BRIEF)



Feature extractor (BRIEF)

The weighted Hamming distance is computed by:

$$WHam(x, y) = \sum_{i=1}^n w_i(b_i(x) \otimes b_i(y))$$

$$b_1 = 0000000001\dots$$

$$b_2 = 0000000011\dots$$

$$b_1 \otimes b_2 = 0000000010\dots$$

$$WHam = 1$$

Feature extractor (BRIEF)

1. Optimization: avoid smoothing kernels

By pre-smoothing the patch, increases the stability and repeatability of the descriptors, but decreases performance.

Feature extractor (BRIEF)

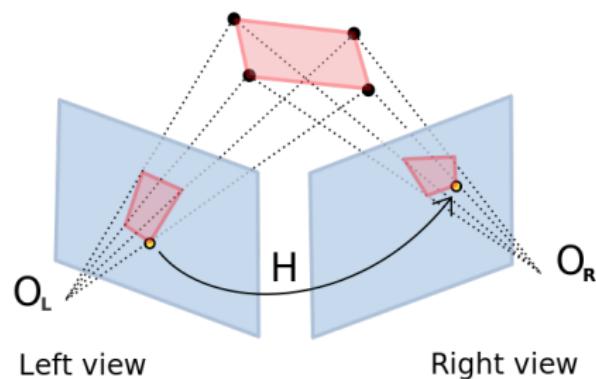
1. Optimization: avoid smoothing kernels

By pre-smoothing the patch, increases the stability and repeatability of the descriptors, but decreases performance.

2. Optimization: avoid Gaussian distribution

Computing the Gaussian distribution can be time consuming.
Simply replaced by a uniform version of JavaScript Math.random().

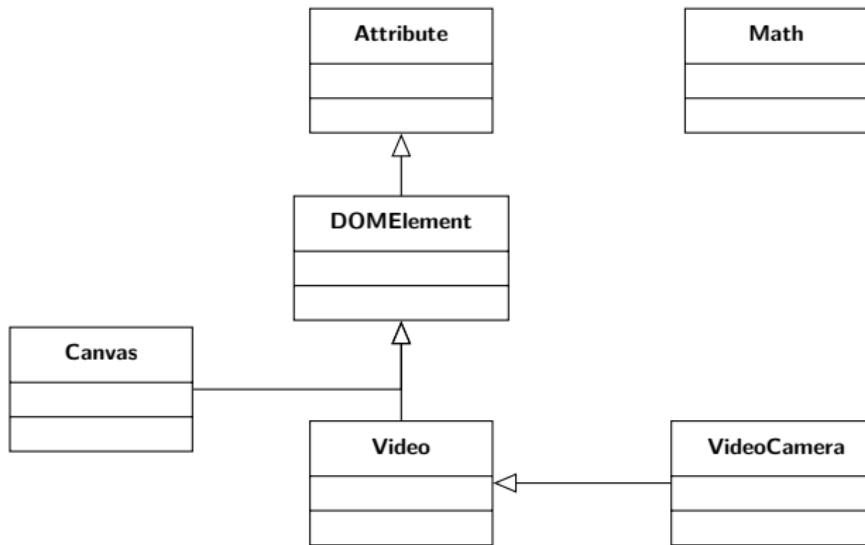
Homography estimation



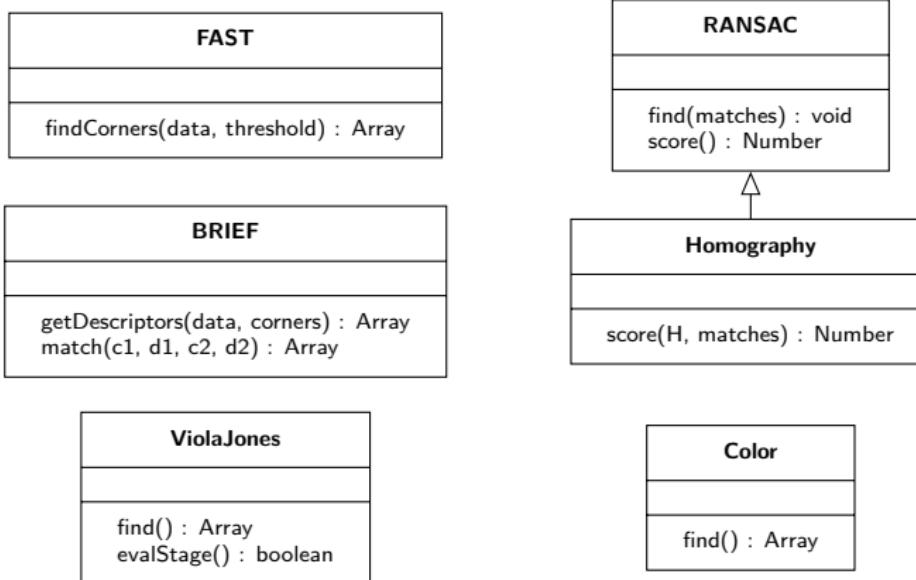
Random sample consensus (RANSAC)

RANSAC (Random Sample Consensus) is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers. It is the most commonly used robust estimation method for homographies.

Library modules - Base classes



Library modules - Visual tracking classes



Outline

1 Introduction

2 Basic concepts

- Web
- Visual tracking
- Visual tracking on the web

3 Tracking library for the web

4 Results and evaluation

- Color tracking algorithm
- Rapid object detection (Viola Jones)
- Markerless tracking algorithm

5 Conclusions

Evaluation methodology

1. Examples

Evaluation methodology

1. Examples

2. Performance

Frames per second (FPS) metric. All tests were executed on Google Chrome browser version 28.0.1500.71, Mac OS X 10.8.3, 2.6 GHz Intel Core i7 16 GB 1600 MHz RAM.



Evaluation methodology

1. Examples

2. Performance

Frames per second (FPS) metric. All tests were executed on Google Chrome browser version 28.0.1500.71, Mac OS X 10.8.3, 2.6 GHz Intel Core i7 16 GB 1600 MHz RAM.

3. Partial occlusion robustness

Examples of how each technique behaves under partial occlusion.

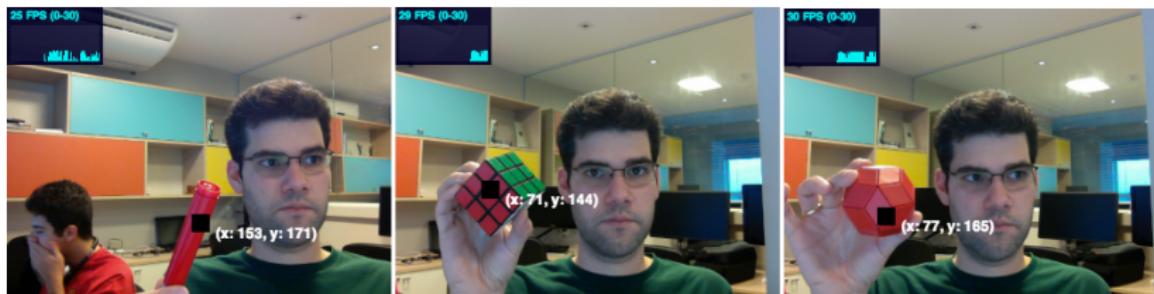
Evaluation methodology

README.md



Color tracking algorithm

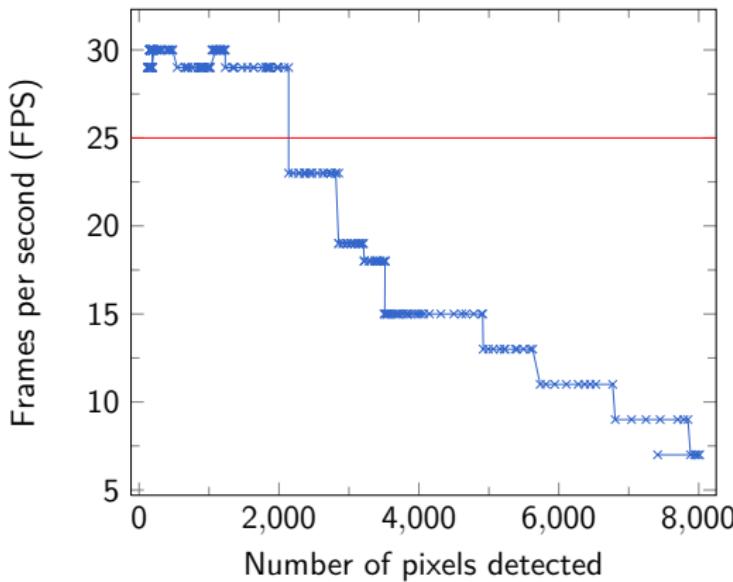
Examples





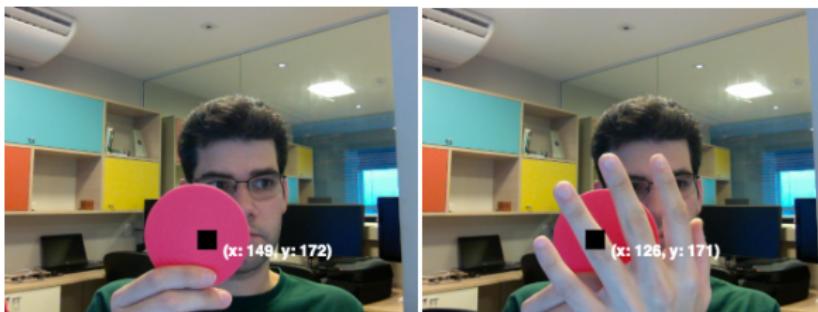
Color tracking algorithm

Performance



Color tracking algorithm

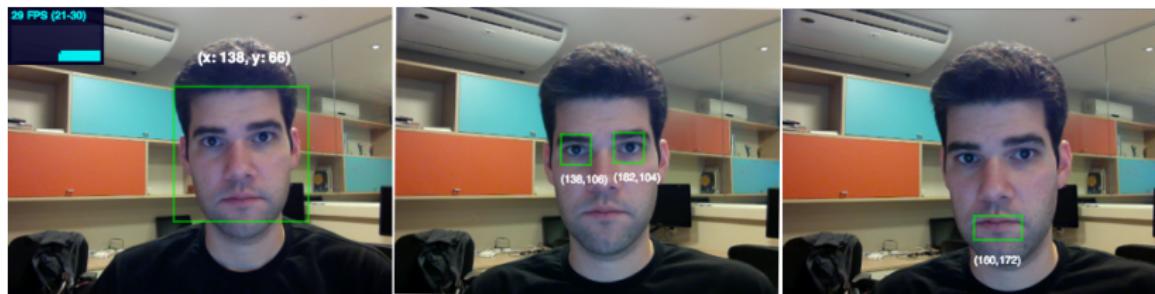
Oclusion robustness





Rapid object detection (Viola Jones)

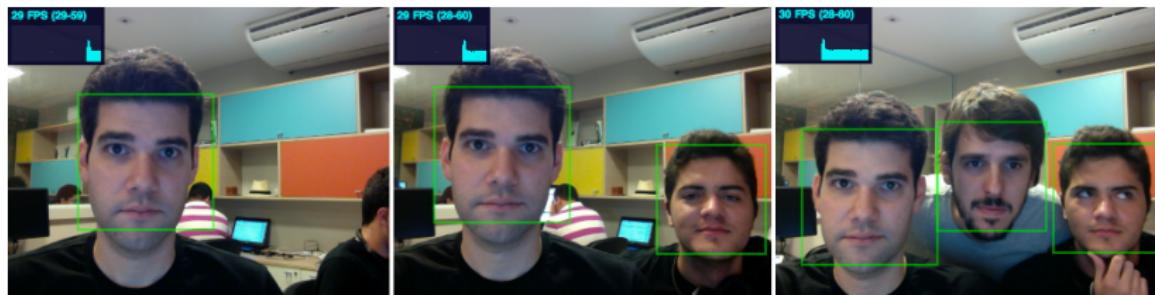
Examples





Rapid object detection (Viola Jones)

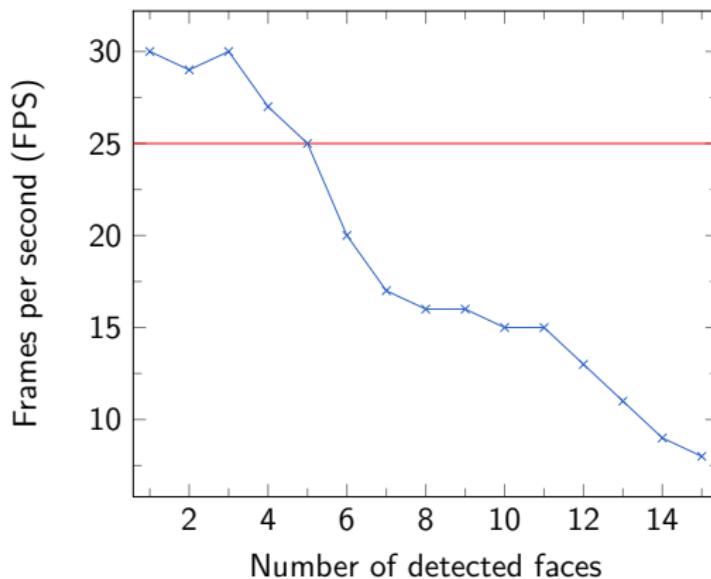
Examples





Rapid object detection (Viola Jones)

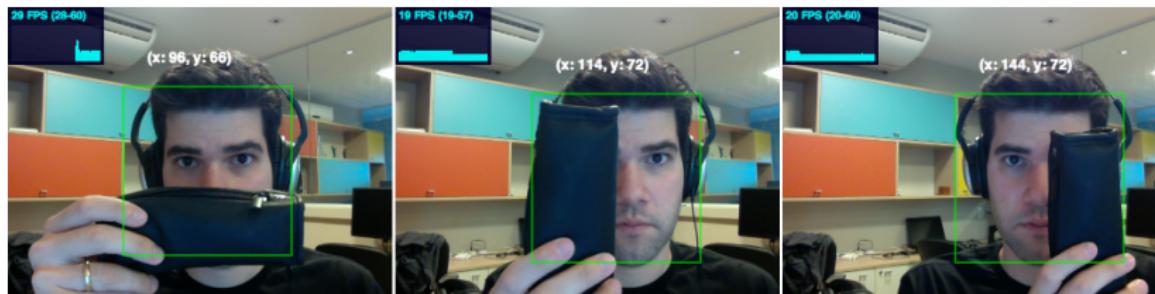
Performance





Rapid object detection (Viola Jones)

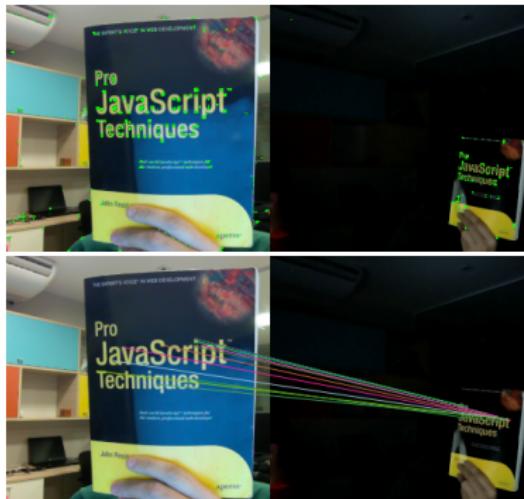
Occlusion robustness



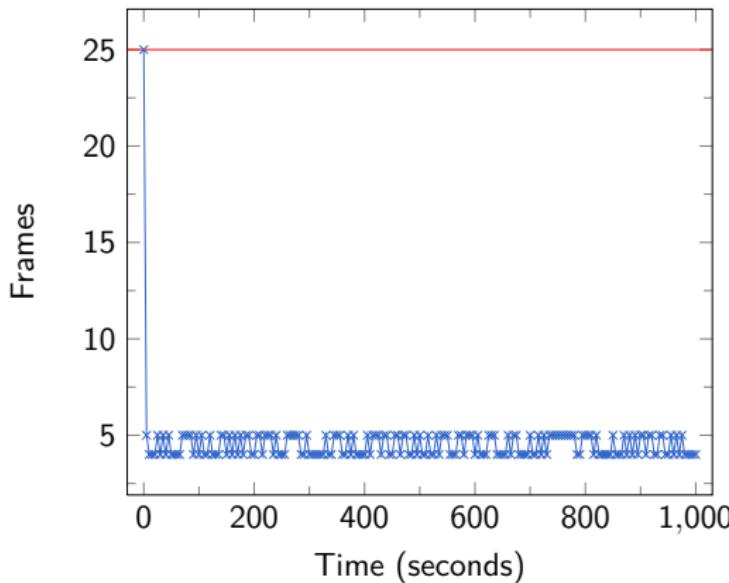


Markerless tracking algorithm

Examples



Performance



Markerless tracking algorithm

Oclusion robustness



Outline

1 Introduction

2 Basic concepts

- Web
- Visual tracking
- Visual tracking on the web

3 Tracking library for the web

4 Results and evaluation

- Color tracking algorithm
- Rapid object detection (Viola Jones)
- Markerless tracking algorithm

5 Conclusions

Benefits of a JavaScript tracking solution

	OSAKit	<i>tracking.js</i> + color	<i>tracking.js</i> + face
Avg frame rate	25 FPS	30 FPS	25 FPS
Avg file size	22 MB	8 KB	187 KB
Avg load time (256 Kbps)	11 min	0.25 seconds	5 seconds

Contributions

- Academic material about web
- Pioneered a tracking library for the web called *tracking.js*
- Optimizations for existing techniques

Future work

- Publication about web browser concepts
- Publication comparing JavaScript tracking solutions with plugin-based and server-side solutions
- Double Exponential Smoothing technique
- Perspective- n -Point problem (PnP) calculation
- Image processing layer capable of transformations and filters



Website

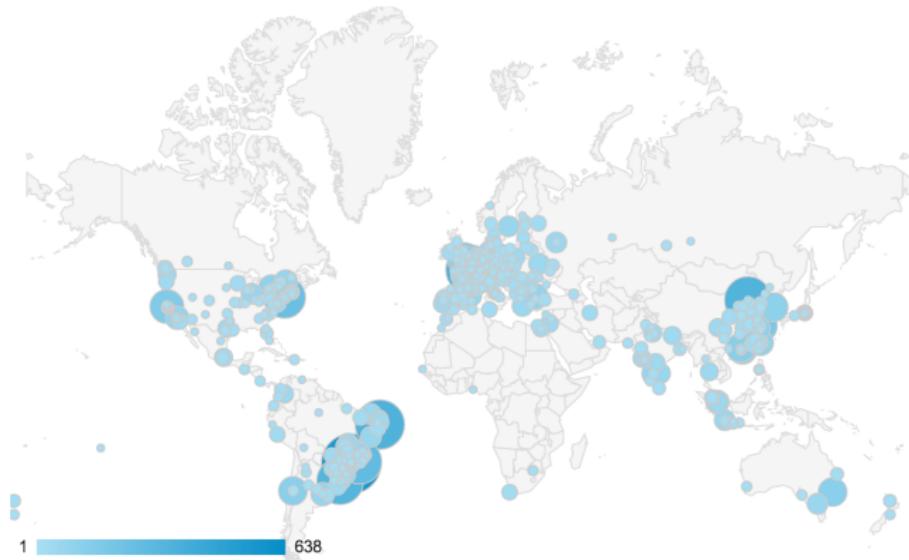
The screenshot shows the homepage of tracking.js. At the top, there are navigation links for "tracking.js", "About", and "Examples". On the right, there are buttons for "Download now" and "Fork on Github". The main title "tracking.js" is prominently displayed in a large, bold font, with the tagline "Change the way you interact with your browser" below it. A video thumbnail in the center shows a person interacting with a tracked object on a screen. Below the video, there are two buttons: "Download now" and "Fork on Github". Further down, there's a section titled "Example" with a code snippet for "Basic Usage". The code uses the `tracking.VideoCamera` constructor to initialize a camera and start tracking objects with color `magenta`.

```
var VideoCamera = new tracking.VideoCamera().render().renderVideoCanvas();
VideoCamera.track([
  type: 'color',
  color: 'magenta',
  onFrame: function(frame) {
    console.log(frame.x, frame.y, frame.x2);
  },
  selectBounds: function() {}
]);
```

Figure : <http://trackingjs.com>



Popularity



Popularity

**David Herman**

@littlecalculist



Following

Spontaneous applause for Eduardo Lundgren's incredible demos of trackingjs.com #braziljs

**Lisa Larson-Kelley**

@lisamarlenyc



Following

A library I wish I had back in the day with Flash -- #javascript #webcam gestures, tracking trackingjs.com buff.ly/1cmLNp2

**HTML5 Weekly**

@HTML5Weekly



Follow

Tracking.js: Webcam/Camera Based Interaction for Web Page Elements - trackingjs.com



Popularity

- 445 tweets mentioning *tracking.js*

Popularity

- 445 tweets mentioning *tracking.js*
- 31,258 website page views

Popularity

- 445 tweets mentioning *tracking.js*
- 31,258 website page views
- Project hosted on Mozilla Developer Network

Popularity

- 445 tweets mentioning *tracking.js*
- 31,258 website page views
- Project hosted on Mozilla Developer Network
- Talk presented on DevFest Brazil - Brazil, 2012

Popularity

- 445 tweets mentioning *tracking.js*
- 31,258 website page views
- Project hosted on Mozilla Developer Network
- Talk presented on DevFest Brazil - Brazil, 2012
- Talk presented on W3C Conference - Brazil, 2012

Popularity

- 445 tweets mentioning *tracking.js*
- 31,258 website page views
- Project hosted on Mozilla Developer Network
- Talk presented on DevFest Brazil - Brazil, 2012
- Talk presented on W3C Conference - Brazil, 2012
- Talk presented on HTML5 Dev Conf - USA, 2013

Popularity

- 445 tweets mentioning *tracking.js*
- 31,258 website page views
- Project hosted on Mozilla Developer Network
- Talk presented on DevFest Brazil - Brazil, 2012
- Talk presented on W3C Conference - Brazil, 2012
- Talk presented on HTML5 Dev Conf - USA, 2013
- Talk accepted on JS.everywhere - USA, 2013

Thank you!