Tracking Library for the Web (tracking.js)

by

Eduardo A. Lundgren Melo

Submitted to the Center for Informatics
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

FEDERAL UNIVERSITY OF PERNAMBUCO

February 2013

Author .....
Center for Informatics
Mar 20, 2013

Certified by .....
Silvio de Barros Melo
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Master Theses

Tracking Library for the Web (tracking.js)

by

Eduardo A. Lundgren Melo

Submitted to the Center for Informatics
on Mar 20, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

In this thesis, I designed and implemented a tracking library for the web aiming to provide a common infrastructure to develop applications and to accelerate the use of those techniques on the web in commercial products. It runs on native web browsers without requiring third-party plugins installation. This involves the use of several modern browser specifications as well as implementation of different computer vision algorithms and techniques into the browser environment. These algorithms can be used to detect and recognize faces, identify objects, track moving objects, etc. The source language of the framework is JavaScript that is the language interpreted by all modern browsers. The majority of interpreted languages have limited computational power when compared to compiled languages, such as C. The computational complexity involved in AR requires highly optimized implementations. Some optimizations are discussed and implemented on this work in order to achieve good results when compared with similar implementations in compiled languages. A series of evaluation tests were made, to determine how effective these techniques were on the web.

Thesis Supervisor: Silvio de Barros Melo
Title: Associate Professor

Acknowledgments

This is the acknowledgements section. You should replace this with your own acknowledgements [16] foo [16].

This master thesis has been examined by a Committee as follows:

Professor Silvio de Barros Melo .....
Thesis Supervisor
Associate Professor

Professor Veronica Teichrieb .....
Chairman, Thesis Committee
Associate Professor

Professor Alvo Dumbledore .....
Member, Thesis Committee
Hogwarts School of Witchcraft and Wizardry

Contents

List of Figures

List of Tables

List of Acronyms

AJAX	Asynchronous JavaScript and XML
BAST	Bug Report Analysis and Search Tool
BTT	Bug Report Tracker Tool
BRN	Bug Report Network
CCB	Change Control Board

Listings

Chapter 1

Introduction

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

1.1 Motivation

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

1.2 Problem Definition

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

1.3 Objectives

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

1.4 Thesis Structure

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Chapter 2

Basic Concepts

2.1 Web

Using concepts from existing hypertext systems, Tim Berners-Lee, computer scientist and at that time employee of CERN, wrote a proposal in March 1989 for what would eventually become the World Wide Web (WWW) [47].

The World Wide Web is a shared information system operating on top of the Internet [47]. Web browsers retrieve content and display from remote web servers using a stateless and anonymous protocol called HyperText Transfer Protocol (HTTP) [47]. Web pages are written using a simple language called HyperText Markup Language (HTML) [47]. They may be augmented with other technologies such as Cascading Style Sheets (CSS) [48], which adds additional layout and style information to the page, and JavaScript (JS) language [31], which allows client-side computation [47]. Client-side refers to operations that are performed by the client in a client–server relationship in a computer network. Typically, a client is a computer application, such as a web browser, that runs on a user's local computer or workstation and connects to a server as necessary. Browsers typically provide other useful features such as bookmarking, history, password management, and accessibility features to accommodate users with disabilities [16].

In the beginning of the web, plain text and images were the most advanced features available on the browsers [47]. In 1994, the World Wide Web Consortium (W3C) was

founded to promote interoperability among web technologies. Companies behind web browser development together with the web community, were able to contribute to the W3C specifications [47]. Today's web is a result of the ongoing efforts of an open web community that helps define these technologies and ensure that they're supported in all web browsers [16]. Those contributions transformed the web in a growing universe of interlinked pages and applications, with videos, photos, interactive content, 3D graphics processed by the Graphics Processing Unit (GPU) [24], and other varieties of features without requiring any third-party plugins installation [19]. The significant reuse of open source components among different browsers and the emergence of extensive web standards have caused the browsers to exhibit "convergent evolution" [16].

The browser main functionality is to present a web resource, by requesting it from the server and displaying it on the browser window [49]. There are four major browsers used today: Internet Explorer, Firefox, Safari and Chrome. Currently, the usage share of Firefox, Safari and Chrome together is nearly 60% [49].

2.1.1 Browser Architecture

Three mature browser implementations were selected and, for each browser, a conceptual architecture was described based on domain knowledge and available documentation. Firefox version 16.0, Safari version 6.0.4 and Chrome version 25.0.1364 were used to derive the reference architecture because they are mature systems, have reasonably large developer communities and user bases, provide good support for web standards, and are entirely open source [47, 16]. The reference architecture for web browsers is shown in Figure 2-1; It comprises eight major subsystems plus the dependencies between them [16]:

1. User interface: includes the address bar, back and forward buttons, bookmarking menu etc. Every part of the browser display except the main window where you see the requested resource [16].

2. Browser engine: an embeddable component that provides a high-level interface

for querying and manipulating the Rendering engine [16, 27].

3. Rendering engine: performs parsing and layout for HTML documents [16, 27].

4. Networking subsystem: used for network calls, like HTTP requests. It has platform independent interface and underneath implementations for each platform [16, 27].

5. JavaScript parser: parses and executes the JavaScript [31] code [16].

6. XML Parser: parse the HTML markup into a parse tree [19], HTML is rather close to XML [27, 19].

7. UI backend: provides drawing and windowing primitives, user interface widgets, and fonts. Underneath it uses the operating system user interface methods [16].

8. Data persistence: stores various data associated with the browsing session on disk, including bookmarks, cookies, and cache [16, 27].

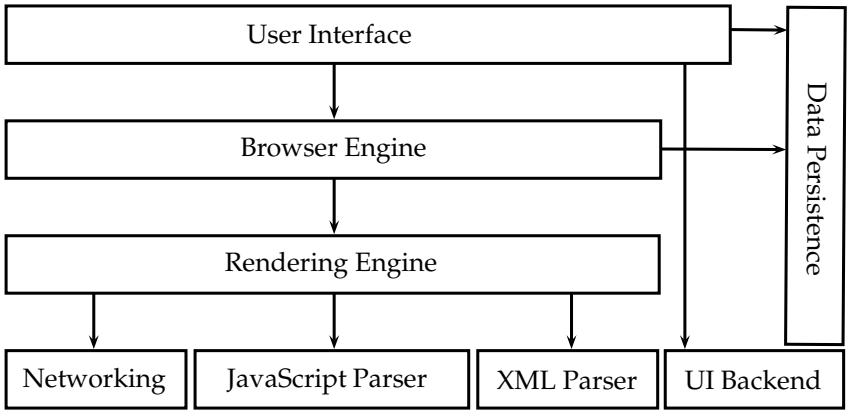

Figure 2-1: Reference architecture for web browsers

Browser subsystem are swappable [16] and could vary for browser vendor, platform or operational system. The browsers mostly differ between different vendors in subsystems (2) the Browser engine, (3) the Rendering engine, and (5) the JavaScript parser [40, 22, 23, 25]. In Firefox, subsystems (2) and (3) are known as Gecko [40, 38], Safari

as WebKit [22, 23] and Chrome uses a fork of WebKit project called Blink [25, 26]. Those browsers subsystems, often called Browser engines, are shown on Figure 2-2.

Another common swappable subsystem is (5) the JavaScript parser. JavaScript [31] is a lightweight, interpreted, object-oriented language with first-class functions, most known as the scripting language for Web pages [38]. The JavaScript standard is ECMAScript [38, 31]. As of 2013, all modern browsers fully support ECMAScript 5.1. Older browsers support at least ECMAScript 3 [38, 31].

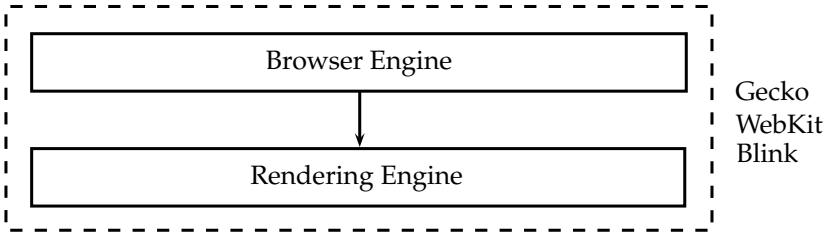

Figure 2-2: Reference architecture for browsers engines

2.1.2 Audio and Video

Audio and video elements were introduced into the browsers by HTML5 [19] specification. Audio and video are HTML5 [19] features that attracts a lot of attention. Often presented as an alternative to Flash [21] in the media, the video element has advantages due to its natural integration with the other layers of the web development stack such as CSS [48] and JavaScript [31] as well as the other HTML elements [47]. The three video formats supported by the three well known browsers cited in this thesis are [47, 27], webm (VP8 Vorbis) [13], mp4 (H.264 AAC) [32] and ogv (Theora Vorbis) [50]. The audio formats available are ogg (Theora Vorbis) [50] and mp4 (H.264 AAC) [32].

Audio and video addition to the browser environment was a good first step for visual tracking applications, on the other hand, the browsers were still not capable of capturing audio and video from the user's camera and microphone, respectively. Audio and video capture has been a limitation of web browsers for a long time [19]. For many years the authors had to rely on browser plugins, such as Flash [21] or

Silverlight [36, 27]. With HTML5 [19], you can now add media to a webpage with just a line or two of code [23]. Browser audio and video elements [19] are shown in Figure 2-3.

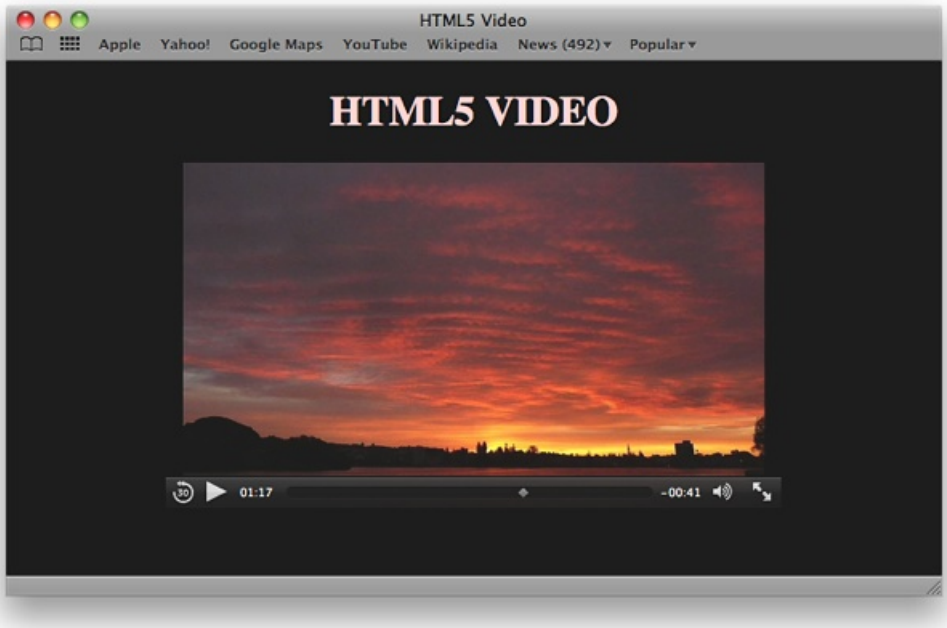

Figure 2-3: Video and audio HTML5 elements [23]

Missing audio and video browser capturing is no longer a problem for the modern browsers cited in this thesis [29, 19]. HTML5 specification [19] has brought a surge of access to device hardware, including Real-time Communication Between Browsers specification (WebRTC) [29] and with Media Capture and Streams specification [11]. Together, they provide a set of HTML5 [19] and JavaScript [31] APIs [29] that allows local media, including audio and video, to be requested from the user platform [47]. With Media Capture and Streams specification [11], the browser can access the camera and microphone input without requiring third-party plugin installation. It's available directly into the browser.

The access camera and microphone hardware access can be used in combination with the HTML5 [19] audio and video elements. An example of camera and microphone capturing using a video element is shown on Listing 2.1.

---

1 <video autoplay></video>

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

Listing 2.1: Capture and display microphone and camera

In Section 2.1.3, a new HTML5 [19] called canvas [8] is introduced. Trough the canvas element [8] the video frames can be read, thus providing access to the raw binary data information of each frame [8, 19]. This raw binary data, also known as array of pixels [15], is extremely useful for AR applications [45]. Since the AR area of study performed on this thesis is based on video tracking, from now on, it will focus on camera and video. There are several integration steps required from capturing video to reading the array of pixels [15]. In order to enable the browser to capture the user camera [11], stream the information into a video element [19], connect the video to a canvas element [8], to finally access the array of pixels of each video frame is a long run. The access flow of raw binary data captured from videos on modern browsers is shown on Figure 2-4 and comprises five major steps [29, 27]:

1. Hardware access: using HTML5 [19] new media capture and streams specification [11], the browser camera and microphone hardware is accessed [29].

2. Streaming: hardware streams audio and video to the browser UI elements [29].

3. UI elements: using HTML5 [19] video element displays the data stream into the browser viewport [47].

4. Raw binary data: using HTML5 [19] canvas element [8] reads the video frames, providing access to the array of pixels.

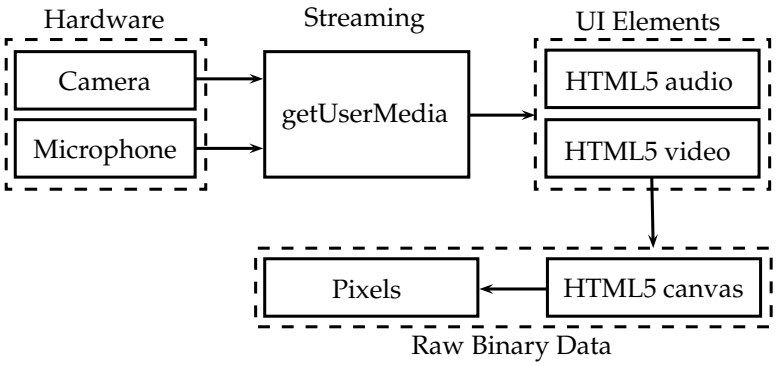

Figure 2-4: Access flow of raw binary data captured from videos on modern browsers

2.1.3 Canvas Element

The canvas [8] is an HTML5 [19] element that provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, art, or other visual images on the fly [8].

Authors should not use the canvas element [8] in a document when a more suitable element is available, e.g. it is inappropriate to use a canvas element [8] to render a page heading. The usage of canvas conveys essentially the same function or purpose as the canvas bitmap. Listing 2.2 shows an example of a basic canvas element [8] HTML markup given a width and height in pixels.

```
1 <canvas width="200" height="200"></canvas>
```

Listing 2.2: The HTML canvas element markup

The canvas [8] is a two-dimensional grid that could be described as a simple computer graphics coordinate system [17]. Normally one unit in the grid corresponds to one pixel on the canvas [8]. The origin of this grid is positioned in the top left corner coordinate $(0, 0)$ [8]. All elements are placed relative to this origin [8]. So the position of the top left corner of the blue square becomes x pixels from the left and y pixels from the top coordinate (x, y) [8]. The canvas coordinate space is shown on Figure 2-5 [39].

For each canvas element [8] a "context" is available. The canvas context provides the drawing context that can be accessed and JavaScript [31] commands can be in-

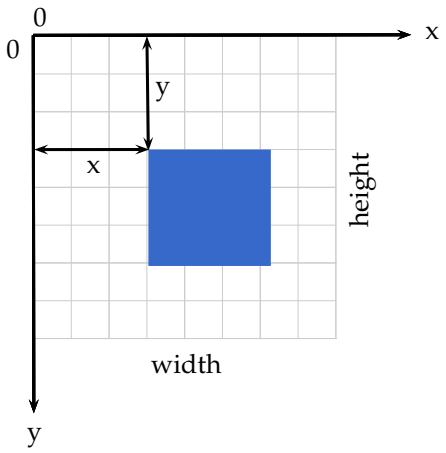

Figure 2-5: The canvas coordinate space

voked to draw or read data [8]. Browsers can implement multiple canvas contexts and the different APIs provide the drawing functionality [8]. Most of the major browsers include the 2D canvas context capabilities. Individual vendors have experimented with their own three-dimensional canvas APIs [8], but none of them have been standardized. The HTML5 [19] specification notes, "A future version of this specification will probably define a 3D context" [8]. Even though 3D context is not available in most part of the major browsers, three-dimensional applications are already being developed based on the 2D canvas context.

Is mandatory the use of the canvas element [8] to develop AR applications on the web, since it's the only way to read video frames array of pixels without any plugin in the browser environment, for more information see Section 2.1.2. Canvas provides APIs to: draw basic shapes, images, videos frames, Bezier [42] and quadratic curves [42, 17]; Apply transformations, translate, rotate and scale; Read raw pixel data etc.

2.1.4 JavaScript Typed Arrays

The JavaScript language [31] is intended to be used within some larger environment, be it a browser, server-side scripts, or similar [16]. JavaScript [31] core language features comprises few major features [39]:

1. Functions and function scope: function is a subprogram that can be called by

code external [39], functions have a scope it references for execution [39].

2. Global objects: refer to objects in the global scope [39], such as general-purpose constructors (*Array, Boolean, Date* etc.), Typed array constructors (*Float32Array, Int32Array, Uint32Array* etc.), Error constructors etc. [39].

3. Statements: consist of keywords used with the appropriate syntax (*function, if...else, block, break, const, continue, debugger etc.*) [39].

4. Operators and keywords: arithmetic operators, bitwise operators, assignment operators, comparison operators, logical operators, string operators, member operators, conditional operator etc. [39].

As web applications become more and more powerful, adding features such as audio and video manipulation, access to raw data using canvas [8] (Section 2.1.2), and so forth, it has become clear that there are times when it would be helpful for JavaScript [31] code to be able to quickly and easily manipulate raw binary data [8, 18]. In the past, this had to be simulated by treating the raw data as a string and using the *charCodeAt()* method to read the bytes from the data buffer [39, 18]. However, this is slow and error-prone [39], due to the need for multiple conversions, especially if the binary data is not actually byte-format data, but, for example, 32-bit integers or floats. Superior, and typed data structures were added to JavaScript specification [31], such as JavaScript typed arrays [39, 31].

JavaScript typed arrays provide a mechanism for accessing raw binary data much more efficiently [39, 18]. This thesis takes advantage of typed arrays in order to achieve acceptable performance and robustness on the web of complex algorithms implementations.

A performance benchmark comparing regular *vs* typed arrays were executed on the three well known open-source browsers, Firefox, Safari and Chrome. The comparison was executed on Mac OS X 10.8.3, 2.6 GHz Intel Core i7 16 GB 1600 MHz RAM. The array types selected were the not strongly typed *Array*; *Float32Array*, represents an array of 32-bit floating point numbers; *Uint8Array*, represents an array of 8-bit unsigned integers [39].

In the benchmark, for each array type a read and a write operation were executed 100,000 times. In order to not compromise benchmark results caused by run-time type conversion [31], the write value used for each array type were proper selected, e.g. *Number* 1.0 was used for regular arrays *Array*, *Number* 1.0 was used for *Float32Array*, and unsigned *Number* 1 for *Uint8Array*. Regular *vs* typed arrays performance benchmark is shown in Figure 2-6 [1].

As conclusion, typed arrays provides faster read and write operations than regular arrays in JavaScript, i.e. 7872 *ops/sec* for unsigned array *vs* 4437 *ops/sec* for regular arrays in Firefox browser, similar behavior is noticeable on Safari and Chrome, thereby float and unsigned arrays are vastly used on complex algorithms implementations on the web.

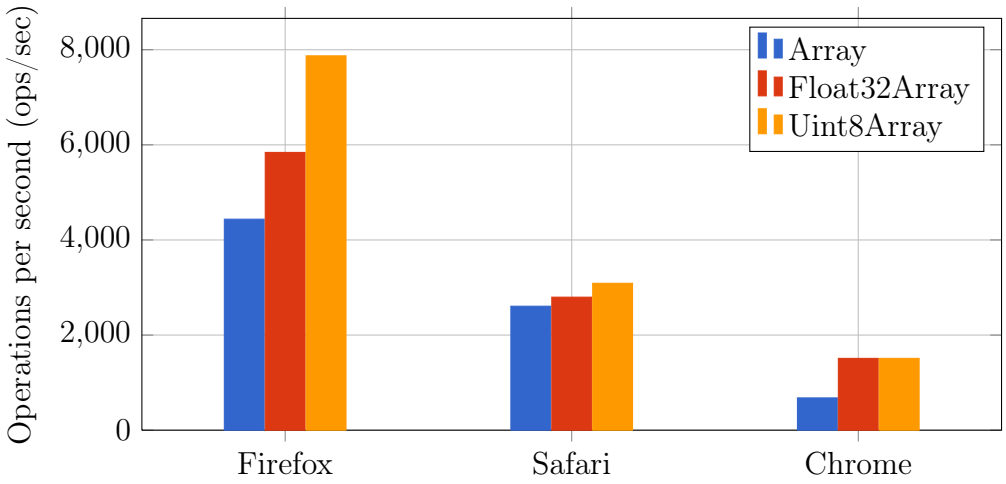

Figure 2-6: Regular *vs* typed arrays performance benchmark

Nevertheless, reading and writing raw binary data [8, 18] using typed arrays only solves part of the problem of manipulating video and images data. The other missing feature was solved by HTML5 [19] canvas element [8], which one important feature is to provide access to the array of pixels of those medias [8, 19]. The raw binary data is used by visual tracking algorithms. For more information see Section 2.1.2.

Videos and images pixels can be drew on a canvas bitmap [8]. Canvas raw binary data can be accessed from the canvas JavaScript [31] API as an object of type *ImageData* [8]. Each object has three properties: width, height and data [8, 39].

The data property is of type *Uint8ClampedArray* [18] that is a one-dimensional array containing the data in RGBA [15] order, as integers in the range 0 to 255. The *Uint8ClampedArray* [18] interface type is specifically used in the definition of the canvas element's 2D API and its structure is similar to the previous shown typed array *Uint8Array* [8, 18].

The *ImageData* data property, or array of pixels, is in row-major order [8, 39], a multidimensional array in linear memory. For example, consider the 2×3 array $\begin{bmatrix} 1 & 2 & 3 \\ 5 & 5 & 6 \end{bmatrix}$, in row-major order it is laid out contiguously in linear memory as $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$. Each array value is represented as integers between 0 and 255 [8, 39], where each four-integer group represents the four color channels of one pixel: red, green, blue and alpha (RGBA). While RGBA [15] is sometimes described as a color space, it is actually simply a use of the RGB [15] color model. This array linear typed structure improves read and write performance, since JavaScript [31] is as an interpreted language [31, 39], previous knowledge of the type results in faster language execution [18, 39]. An example of the canvas [8] image data array of pixels is shown on Figure 2-7.

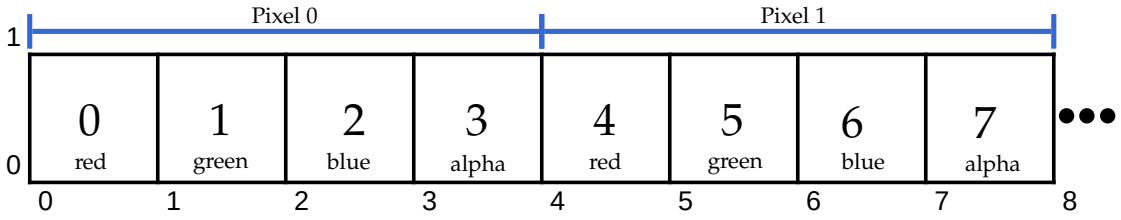

Figure 2-7: The canvas image data array of pixels

2.2 Augmented Reality

Imagine a technology in which you could see more than others see, hear more than others hear, and even touch things that others cannot [34]. A technology able to perceive computational elements and objects within our real world experience that help us in our daily activities, while interacting almost unconsciously through mere

gestures and speech [34, 45].

With such technology, mechanics could see instructions what to do next when repairing an unknown piece of equipment, surgeons could take advantage of augmented reality while performing surgery on them and we could read reviews for each restaurant in the street we're walking in on the way to work [34].

Augmented reality (AR) is this technology to create a "next generation, reality-based interface" [34] and is moving from laboratories around the world into various industries and consumer markets. AR supplements the real world with virtual (computer-generated) objects that appear to coexist in the same space as the real world. AR was recognized as an emerging technology of 2007 [34], and with today's smart-phones and AR browsers we are starting to embrace this very new and exciting kind of human-computer interaction. Three important aspects of an AR system [34, 2]:

1. Combines real and virtual objects in a real environment [34, 2].
2. Aligns real and virtual objects with each other [34, 2].
3. Runs interactively, in three dimensions, and in real time [34, 2].

On the reality-virtuality *continuum* by Milgram and Ki [37], Augmented Reality (AR) is one part of the general area of mixed reality. Both virtual environments (or virtual reality) and augmented virtuality, in which real objects are added to virtual ones, replace the surrounding environment by a virtual one. In contrast, AR provides local virtuality. The reality-virtuality *continuum* is shown on Figure 2-8.

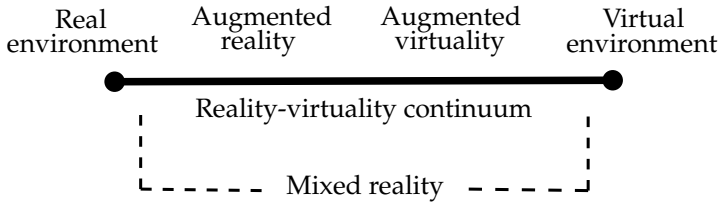

Figure 2-8: Reality-virtuality continuum [2]

The first AR prototypes, shown in Figure 2-9, created by computer graphics pioneer Ivan Sutherland and his students at Harvard University and the University of

Utah, appeared in the 1960s and used a see-through to present 3D graphics [2]. It took until the early 1990s before the term "augmented reality" was proposed by Caudell and Mizell [2], scientists at Boeing Corporation who were developing an experimental AR system to help workers put together wiring harnesses [2].

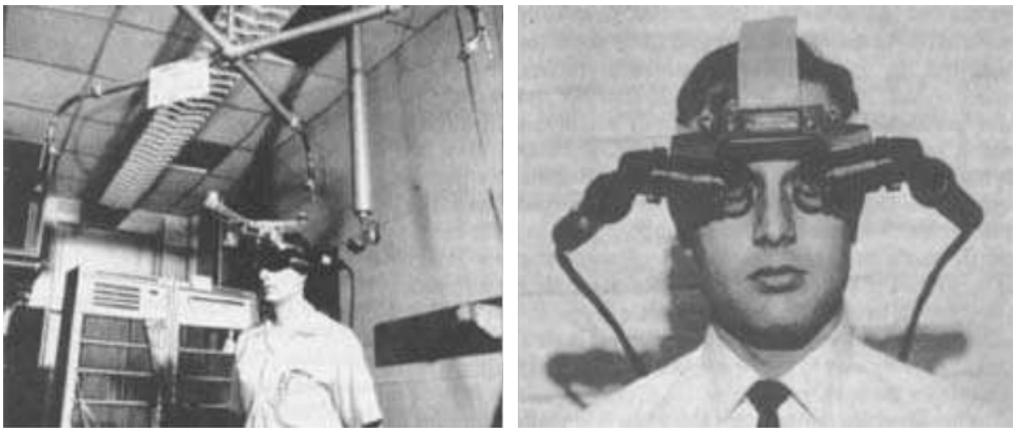

Figure 2-9: The world's first head-mounted display with the "Sword of Damocles"

By the late 1990s, as AR became a distinct field of research [34]. Displays, trackers, graphics computers and software remain essential in many AR experiences [34, 2]. There are several displays devices that could be used on AR applications, the major ones could be divided as follows [2]:

1. Visual display: video see-through, optical see-through, projective and monitor.

   (a) Video see-through: is the closest to virtual reality, where the virtual environment is replaced by a video feed of reality and the AR is placed on the video frames [2].

   (b) Optical see-through: systems combine computer-generated imagery with holographic optical elements (HOEs), providing a "through the glasses" image of the real world [2].

   (c) Projective: these displays have the advantage that they do not require special eye-wear, it projects the information into a surface, wall or even the human palm [2].

(d) Monitor: the use of a regular computer monitor has become popular as a visual display since AR techniques are now available on the [2].

2. Display positioning: head-worn, hand-held and spatial [2].

   (a) Head-worn: is a visual displays attached to the head include the video or optical see-through head-mounted display (HMD), virtual retinal display (VRD), and head-mounted projective display (HMPD) [2]. Google Inc. released a first-class optical HDM called Project Glass shown on Figure 2-10 [28].

Figure 2-10: Optical head-mounted display Google Glass [28] *(Photo credit: CBS)*

   (b) Hand-held: includes hand-held video or optical see-through displays as well as hand-held projectors, it is currently the best choice to introduce AR to a mass market due to low production costs and ease of use since it may be based on existing consumer products, such as mobile phones [2].

   (c) Spatial: those are displays that could placed statically within the environment and include screen-based video see-through displays [2].

In this thesis, it was designed and implemented tracking library for the web aiming to provide a common infrastructure to develop applications and to accelerate the use of those techniques on the web in commercial products. It runs on native web browsers without requiring third-party plugins installation, therefore any browser ready device

can eventually use the proposed cross-platform code base to develop AR applications. In order to develop for different devices different skills are required, since APIs and programming languages may differ between them [39, 31]. In the current day, the most popular devices, such as smart phones, tablets, computers, notebooks and HDM (i.e. Google Glass [28]) [2] are browser ready [19]. They all could benefit from a reusable, cross-platform framework for AR applications.

2.3 Tracking

Tracking an object in a video sequence means continuously identifying its location when either the object or the camera are moving [35]. There are a variety of approaches, depending on the type of object, the degrees of freedom of the object and the camera, and the target application [35, 45]. 2D tracking typically aims at following the image projection of objects or parts of objects whose 3D displacement results in a motion that can be modeled as a 2D transformation [35].

One of the major responsibility of a tracking algorithms is to extract information that allows superposing computer generated images on real scenes [35]. Many potential Augmented Reality (AR) applications have been explored, such as medical visualization, maintenance and repair, annotation, entertainment, aircraft navigation and targeting [35, 34]. The objects in the real and virtual worlds must be properly aligned and the system latency should also be low, lest the illusion that the two worlds coexist be compromised. [35] Due to the importance of tracking in AR, this area owns the biggest number of publications and challenge requests on the International Symposium on Mixed and Augmented Reality (ISMAR) [52].

There are a variety of ways to track a real environment: e.g. global positioning system (GPS) [34], mechanical trackers [34], magnetic trackers [34], ultrasonic trackers [34] etc., but they all have their weaknesses [34]. Mechanical trackers are accurate enough, although they tether the user to a limited working. Magnetic trackers are vulnerable to distortions by metal in the environment. Ultrasonic trackers suffer from noise and tend to be inaccurate at long ranges because of variations in the ambient

temperature [35]. By contrast, vision has the potential to yield non-invasive, accurate and low-cost solutions to this problem, provided that one is willing to invest the effort required to develop sufficiently robust algorithms [35]. Bringing video tracking based techniques on the web are the focus of this work [35]. In some cases, it is acceptable to add fiducials [7], such as special markers, to the scene or target object [7, 35]. The addition of fiducials in the scene, also called landmarks or markers [7], greatly helps since they constitute image features easy to extract, and they provide reliable, easy to exploit measurements for the pose estimation [7, 35]. It is therefore much more desirable to rely on naturally present features, such as edges, corners, or texture [35]. The latter, makes tracking far more difficult because finding and following feature points or edges can be difficult [35]. There are too few of them on many typical objects. Section 2.4 details markerless tracking techniques, main technique implemented on the web by this thesis.

2.4 Markerless Tracking Techniques

Markerless Augmented Reality (MAR) systems integrate virtual objects into a real environment in real-time, enhancing user's perception of and interaction with the world. This approach is not based on the use of traditional artificial markers that need to be placed in the world to be tracked by the system in order to calculate their position and orientation [45], instead, MAR relies on naturally present features in order to position virtual objects. Tracking techniques become more complex in MAR systems when compared to marker techniques. MAR can be classified in two major types: Model based and Structure from Motion (SfM) based. The online monocular MAR taxonomy is shown on Figure 2-11.

2.4.1 Model Based

Model-based trackers usually construct their models based on lines or edges in the model. Edges are the most frequently used features as they are computationally efficient to find and robust to changes in lighting [52]. Texture is another useful

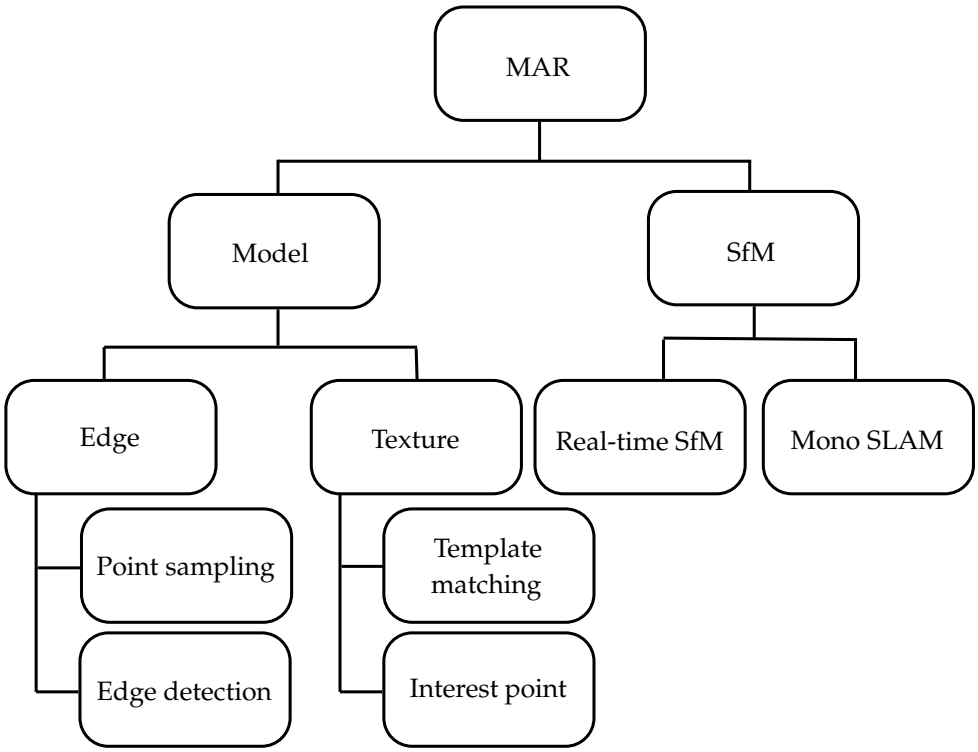

Figure 2-11: Online monocular Markerless Augmented Reality taxonomy

feature for constructing models. The advantage of using a model based approach is the possibility of interaction between real and virtual worlds [45], like occlusion and collision. In order to accomplish such types of interaction, the application exploits the fact that the real object pose is known and its structure is described by the 3D model [45]. Model based approaches require an a priori knowledge about the real scene [45]. Due to the offline model acquiring process, these techniques are, in general, not totally online and cannot be used in unprepared environments [45]. Furthermore, tracker initialization is usually done manually or requires a prior training in order to be automatic [45]. Model based techniques can be classified in three categories:

1. Edge based: the camera pose is estimated by matching a wireframe 3D model of an object with the real world image edge information. This matching is achieved by projecting the model onto the image and minimizing the displacement between the projected model and the imaged object [45, 35]. In edge based methods, the initialization is done manually [45]. Once the first pose is

estimated, it is used to project the model onto the next frame. Assuming that camera displacement between consecutive frames is relatively small, using the previous pose estimation to predict the next pose will not harm the matching process [45]. Edge based methods usually do not support fast camera movements, since the projected model will be too far from its correct location [45]. Edge based methods can be divided in two subcategories [45]:

(a) Point sampling: sample some control points along the edges of the wire-frame 3D model and compare their projections with strong gradients present in the image [45]. Edge based by point sampling is shown on Figure 2-12.

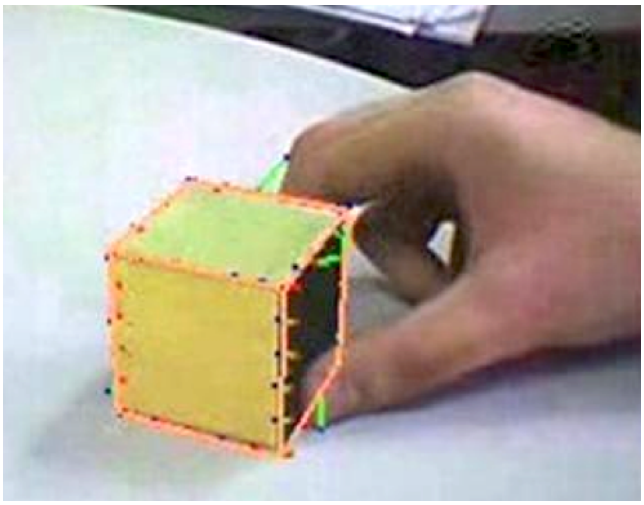

Figure 2-12: Edge based by point sampling

(b) Edge detection: detect explicit edges on the image and match them with the model projection [45]. Edge based by edge detection is shown on Figure 2-13.

2. Texture based: takes into account texture information presented in images [45, 35]. Texture based technique could be divided in two sub-categories: (a) template-matching [45, 35] and (b) interest point [45, 35]. The technique implemented in this thesis is texture based, furthermore, a much detailed description is presented:

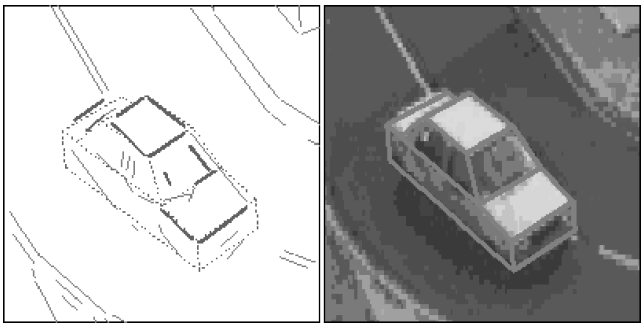

Figure 2-13: Edge based by edge detection [45]

(a) Template matching: is based on global information, this subcategory lies in its ability to treat complex patterns that would be difficult to model by local features [45, 35]. These techniques are also called sum-of-square-difference (SSD) [45, 43], as they consist in minimizing the difference between a region of the image and a reference template [45, 35]. Texture based by template matching is shown on Figure 2-14.

Figure 2-14: Texture based by template matching

(b) Interest point: takes into account localized features [45]. As a result, this subcategory is less computer-intensive than template matching approach [45, 43], this characteristic makes interest point technique a good choice for AR applications on the web. Texture based by interest points is shown on Figure 2-15.

Figure 2-15: Texture based by interest points

The interest point based tracking solution proposed, uses Features from Accelerated Segment Test (FAST) [44] to extracts feature points using corner detection. This technique is detailed on Section 3.4. Binary Robust Independent Elementary Features (BRIEF) [6] is used as an efficient feature point descriptor (also known as feature matching), detailed on Section 3.5. Both techniques combined results in a high performance marker-less tracking solution which fits the native web needs due to the existing browsers performance limitations [6, 43].

2.4.2 Structure from Motion (SfM)

One interesting aspect of SfM based techniques are that they are mainly online, since they do not require any previous offline learning phase [45]. Therefore, it is possible to reconstruct a totally unknown environment on the fly. As a drawback, SfM approaches are often very complex and have some constraints related to their real-time nature [45]. Instead of relying on previously obtained information about the scene to be tracked, some MAR techniques estimate the camera displacement without any a priori knowledge about the environment. These methods are also able to retrieve the structure of the scene in real-time, with different levels of detail, depending on the approach used [45].

SfM based systems have advantages, since they are capable of continuously tracking the camera in unknown scenes [45]. The focus of this thesis is provide texture

based techniques on the web, therefore SfM techniques are not detailed in this thesis. Model based systems requires lower computational complexity, ideal to the web environment. Real time tracking using Structure from Motion (SfM) techniques [45] is shown on Figure 2-16.

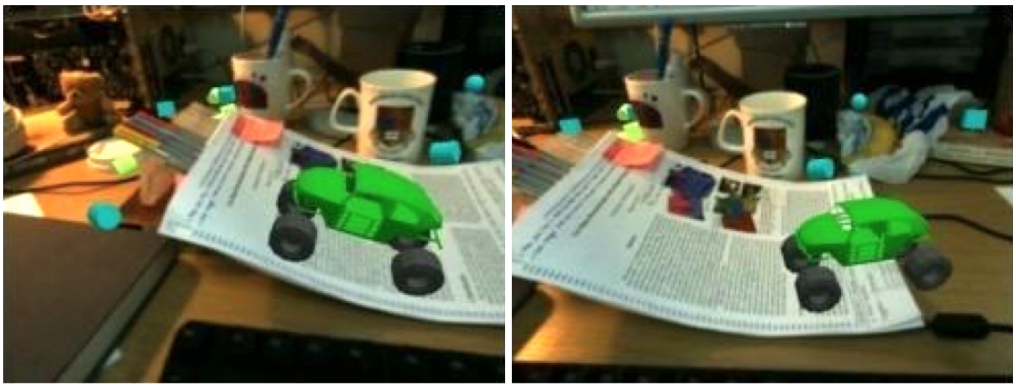

Figure 2-16: Real time tracking using Structure from Motion (SfM) techniques

Chapter 3

Tracking Library for the Web (tracking.js)

3.1 Introduction

The desktop platform is the target environment most commonly addressed when developing Augmented Reality (AR) systems. However, depending on the requirements of an AR application, the use of different execution platforms may be necessary. If the system has to be published to several users, the web platform shows to be more adequate, where the application is executed through the Internet in a web browser [41]. The use of markerless tracking, which is based on natural features of the scene, has also been gaining more space on web targeted AR applications for advertising. The media used in this kind of application needs to be as appealing as possible in order to catch consumers' attention. Markerless tracking satisfies this requirement, since the idea of having a real scene augmented with virtual objects without any artificial elements such as markers added to the environment is very attractive [41]. In addition, the product being advertised can be tracked and augmented with virtual elements [41]. Browsers are evolving very fast when compared to the the previous decade [19]. JavaScript language [31, 39] wasn't prepared to handle typed data structures [18] able to manipulate raw binary data safely [8], all the computational complexity required by AR algorithms was too much for that growing environment. Browsers

weren't able to capture audio and video [11, 29] natively, without plugin installation [21], an essential feature for AR applications. This reality has changed, this involves the use of several modern browser specifications [19, 47] as well as implementation of different computer vision algorithms and techniques into the browser environment taking advantage of all those modern APIs [19, 47].

In this context, this thesis aims to present the implementation and evaluation of a solution regarding tracking techniques for web targeted AR. These algorithms can be used to detect and recognize faces, identify objects, track moving objects etc. The solution is called *tracking.js*. Some optimizations are discussed and implemented on this work in order to achieve good results when compared with similar implementations in compiled languages.

3.1.1 Library Modules

The proposed library is divided in modules in order to allow extension and addition of new features, such as new RA techniques or math utils. For a better understanding of the library architecture, the current implementation is divided in two packages separating Base from AR classes. Base classes modules are shown in Figure 3-1 and AR classes in Figure 3-2.

To develop AR applications using only raw JavaScript [31] APIs [39] could be too verbose and complex, e.g. capturing users' camera and reading its array of pixels. The big amount of steps required for a simple task makes web developers life hard when the goal is to achieve complex implementations. Some level of encapsulation is needed in order to simplify development. The proposed library provides encapsulation for common tasks on the web platform.

The two main available packages splits Base from AR classes. Furthermore, each class of those packages are described. Let's start with the base classes:

1. Math: provides common math utilities optimized for the web, such as geometry, linear algebra [17], hamming operations etc. Typed arrays [18] are used in order to optimize performance, see subsection 2.1.4 for more information about typed

arrays.

2. Attribute: allows developers to add attributes to any class through an augmentable Attribute interface. The interface adds get and set methods to your class to retrieve and store attribute values, as well as support for change events that can be used to listen for changes in attribute values.

3. DOMElement: provides a way to create, and manipulate HTML [19] DOM nodes [47]. Each DOMElement instance represents an underlying DOM node [47]. In addition to wrapping the basic DOM API [47] and handling cross browser issues, Nodes provide convenient methods for managing styles, subscribing to events, etc.

4. Canvas: provides an utility class to create, and manipulate HTML5 [19] canvas element [8]. Each Canvas instance represents an underlying canvas DOM node [8]. In addition to wrapping the basic DOM API [47], also provides methods to extract via *getImageData* method, to loop via *forEach* method, and to set the canvas array of pixels via *setImageData* method.

5. Video: provides an utility class to create, and manipulate HTML5 [19] video element. Each Video instance represents an underlying video DOM node [8]. In addition to wrapping the basic DOM API [47], also provides methods to *play*, *pause* and register tracker algorithms via *track* method. See subsection 2.1.2 for more information about video element.

6. VideoCamera: extends all functionalities from Video class with the addition of capturing the user camera via *capture* method. The underlying implementation uses WebRTC [29] and Media Capture and Streams [11] specifications.

Visual tracking classes includes several computer vision algorithms, such as FAST [43], BRIEF [6] implementations, homography estimation and others. As the library grows, many other computer vision algorithms are going to be added to the library, such as 3D pose calculation.

1. FAST: provides an implementation of Features from Accelerated Segment Test (FAST) [44] for interest points detection via *findCorners(data, threshold)* method, where *data* is the *ImageData* of the canvas [8] frame. It also dependents on a *threshold* argument. The pixel at p, see Figure 3-6, is the centre of a candidate corner and they are classified if brighter than p by more than the *threshold*.

2. BRIEF: provides an implementation of Binary Robust Independent Elementary Features (BRIEF) [6] for feature description via *getDescriptors(data, corners)* method and matching via *match(c1, d1, c2, d2)*, where *data* is an *ImageData*, and $c1$ and $c2$ are the found corners array return by *FAST.findCorners* method and $d1$ and $d2$ are feature descriptors array return by *BRIEF.getDescriptors* method.

3. RANSAC: provides an interface used to achieve robust estimation method for homographies and camera pose. There are two available estimation methods implemented that inherits from RANSAC [17], Homography and Pose.

4. Homography: provides an API to estimate a homography matrix H between images by finding feature correspondences in those images.

5. Pose: TODO.

6. ViolaJones: TODO.

7. Color: TODO.

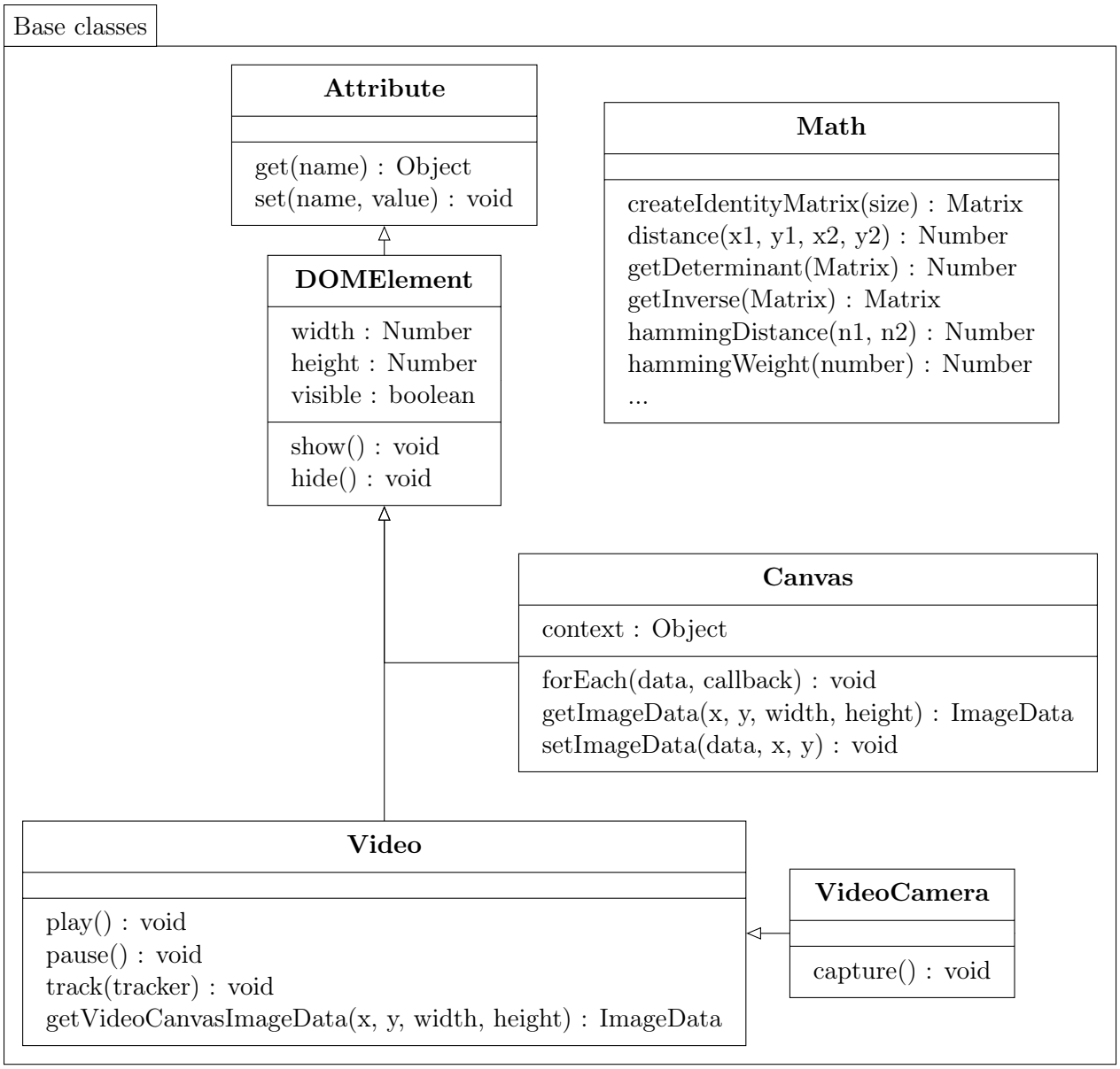

Figure 3-1: Base classes of tracking.js library

3.2 Related Work

There are not many web based RA solutions available and registered in the literature. The ones available are mainly focused on fiducial markers [7], such as FLARToolKit [51] and JSARToolkit [33], they both are ports of ARToolKit [20]. ARToolKit is a desktop library which is useful to make vision-based Augmented Reality applications [20]. The Metaio company developed Unifeye Viewer [30], a proprietary plug-in for

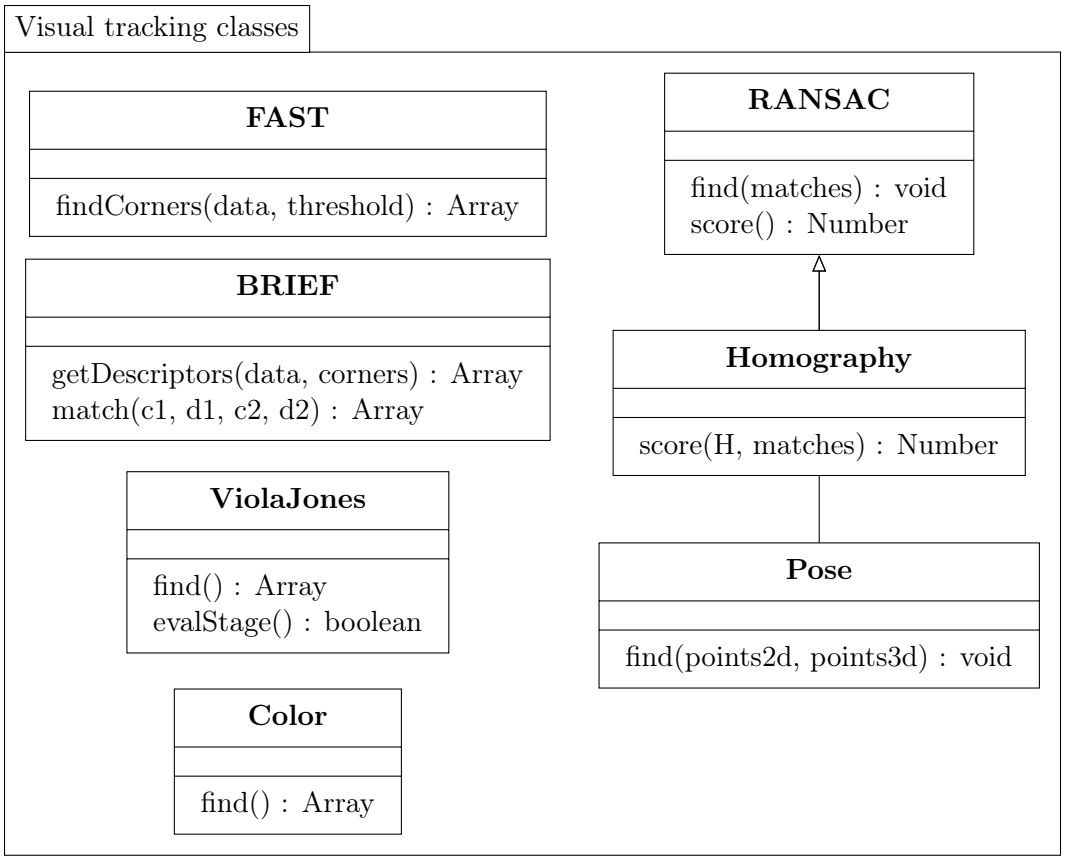

Figure 3-2: Visual tracking classes of tracking.js library

Flash [21] that allows the utilization of markerless AR applications on the web. In order to run Flash [21] based applications, the instalation of its plugin is required. Third-party plugins, such as Flash [21], are in decadency on modern and mobile web browsers, instead JavaScript [31] based solutions are preffered, since they can run in any modern browser without required any user effort of installing external software. Some smart-phones doesn't even support Flash [21] plugin into their browsers, e.g. Safari for mobile [22] is one example of a mobile browser that have banned Flash [21].

1. FLARToolKit: is a port of the well-known ARToolKit [20] marker tracking library to ActionScript [21], which is the language utilized in the development of Flash [21] applications for the web. This was the first initiative towards AR solutions for the web [41]. Using FLARToolKit [51], is possible to develop AR applications that runs on client's browser. A marker based AR example for the web, developed for a marketing campaing of General Electric's company using

FLARToolKit, is shown on Figure 3-3.

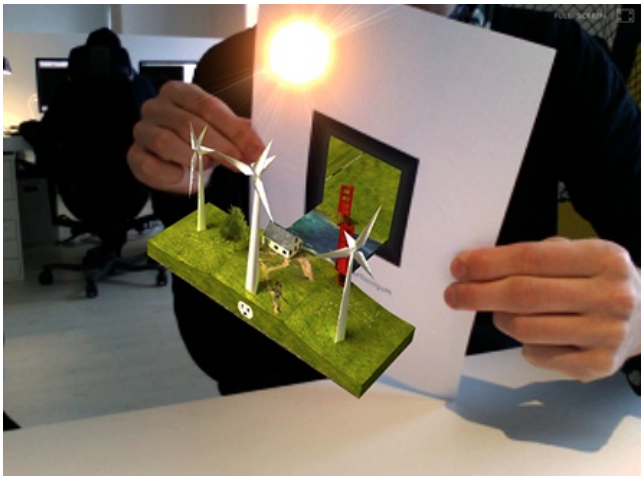

Figure 3-3: Marker based AR for the web using FLARToolKit

2. JSARToolkit: is a JavaScript [31] port of FLARToolKit [51], operating on canvas images [8] and video element [19] contents, provides another marker tracking library. This was the first, open-source, JavaScript [31] based, AR solution available for the web. A marker based AR example for the web using JSARToolKit is shown on Figure 3-4.

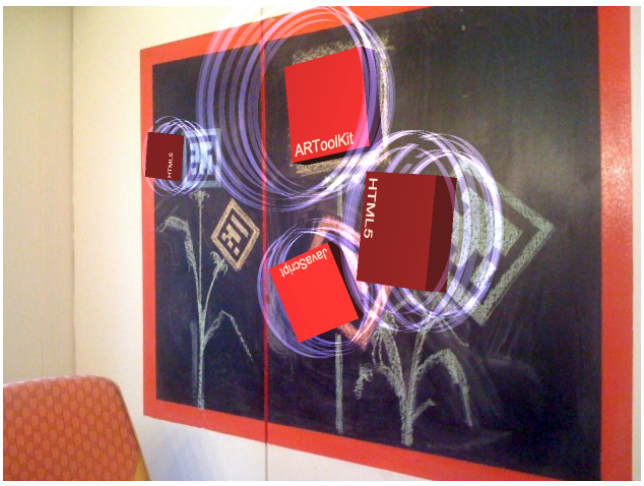

Figure 3-4: Marker based AR for the web using JSARToolKit

3. Unifeye Viewer: from Metaio company, offers a robust markerless tracking solution for the web. Unifeye [30] also depends on Flash [21] plugin in order to

run on web browsers. A similar example of General Electric's marker based solution, this time markerless based, is shown on Figure 3-5. Note that the 3D image is projected over a magazine cover instead of a fiducial marker [7].

Figure 3-5: Marker based AR for the web using JSARToolKit

There is a disadvantage of using marker based augmented reality. Depend on a artificial marker in order to augment the scene with virtual elements is counterintuitive. Commonly, web applications are utilized by novice users that do not have sufficient technical knowledge to perform manual setup, such as print fiducial markers or perform manual initialization for the tracking. FLARToolKit [51] and JSARToolkit [33] are both marker based techniques, using Flash [21] and JavaScript [31], respectively. FLARToolKit has one more issue which is dependency on Flash [21] plugin installation. Unifeye Viewer by Metaio, was the only existing solution that provided markerless tracking for the web, although it uses Flash [21], exlcuding it from a potential competitor of *tracking.js*. Markerless tracking techniques does not depend on any artificial marker or advanced user initialization. The space on web targeted AR applications for advertising is gaining more space and the media used in this kind of application needs to be as appealing as possible [41]. Making markerless tracking a suitable technique to such applications.

The solution proposed in this thesis, *tracking.js*, provides the first known, open-source, markerless tracking solution for the web that runs enterely in JavaScript [31]

and HTML5 [19].

3.3 Markerless Tracking Algorithm

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

3.4 Interest Point Detection

This technique rely on matching individual features across images and are therefore easy to robustify against partial occlusions or matching errors. Illumination invariance is also simple to achieve. Feature points detection is used as the first step of many vision tasks such as tracking, localization, image matching and recognition. In this article we call "feature" or "keypoint" to refer to a point of interest in two dimensions.

For each frame, the object features are matched by localizing feature templates in search windows around hypothesized locations [35]. The method to extract feature points suggested in Features from Accelerated Segment Test (FAST) [44]. FAST [43] hypothesizes the matches using corner detection. A large number of corner detectors exist in the literature. However, we have a strong interest in realtime frame rate applications which computational resources are required requisites. The approach proposed by FAST [43] allows the detector to produce a suite of high-speed detectors which we currently use for real-time tracking and AR label placement [6]. In particular, it is still true that when processing live video streams at full frame rate, existing feature detectors leave little if any time for further processing, even despite the consequences of Moore's Law [44].

To show that speed can been obtained without necessarily sacrificing the quality of the feature detector, in Chapter 4, we compare our detector, to a variety of well-known detectors. A number of the detectors described below compute a corner response, (1) Edge based corner detectors, corresponds to the boundary between two regions; (2) Greylevel derivative based detectors, the assumption that corners exist along edges is an inadequate model for patches of texture and point like features, and is difficult to

use at junctions. Therefore a large number of detectors operate directly on greylevel images without requiring edge detection; and (3) Direct greylevel detectors, Another major class of corner detectors work by examining a small patch of an image to see if it "looks" like a corner [44].

The thesis choice was (3) Direct greylevel detectors. It works by testing a small patch of an image to see if it could be a corner. The detector is evaluated using a circle surrounding the candidate pixel, the test is based on whether the concentric contiguous arcs around the pixel are significantly different from the central pixel p [44]. To classify p as a corner should exists a set of n contiguous pixels in the circle which are all brighter than the intensity of the candidate pixel $I_p + t$ (threshold), or all darker than $I_p - t$ [44].

The number of contiguous tested pixels could vary accordingly [44], being more common to be FAST-12 or FAST-9. Empirically, FAST-9 showed to have a good repeatability and a better efficiency on the web. The repeatability of found feature points is also important because determines whether the technique is useful in a real-world application.

This detector in itself exhibits high performance, but there are several weaknesses: (1) This high-speed test does not reject as many candidates; (2) The efficiency of the detector will depend on the ordering of the questions and the distribution of corner appearances; and (3) Multiple features are detected adjacent to one another [44].

On Figure 3-6, the highlighted squares are the pixels used in the corner detection. The pixel at p is the central pixel. The arc is indicating that the dashed line passes through FAST-n, let n be 9 or 12 contiguous pixels which are brighter or darker than p [44].

3.5 Interest Point Description and Matching

To estimate motion, one can then match sets of interest points $\{m_i\}$ and $\{m'_j\}$ extracted from two images taken from similar, and often successive, viewpoints. A classical procedure [6] runs as follows. For each point $\{m_i\}$ in the first image, search

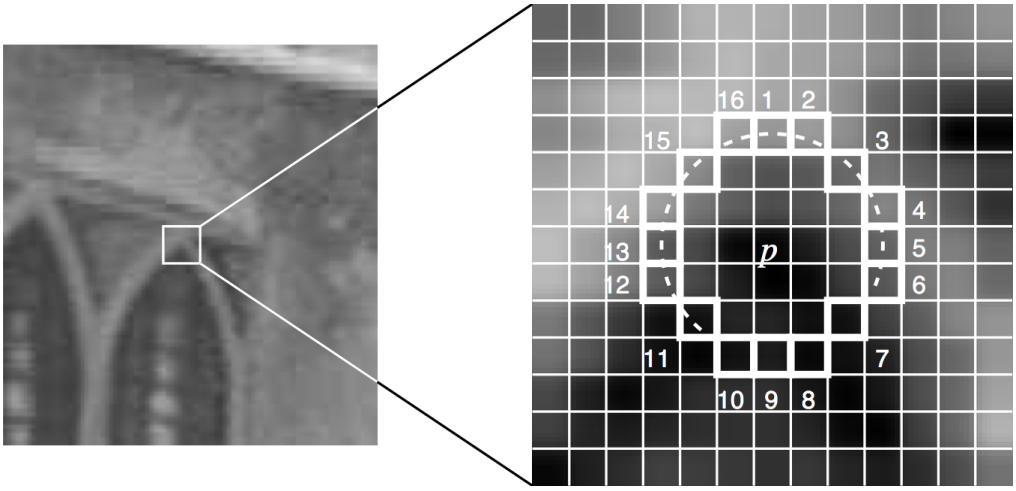

Figure 3-6: Point segment test corner detection in an image patch [28]

in a region of the second image around location $\{m_i\}$ for point $\{m'_j\}$. The search is based on the similarity of the local image windows, also knowns as kernel windows, centered on the points, which strongly characterizes the points when the images are sufficiently close [35].

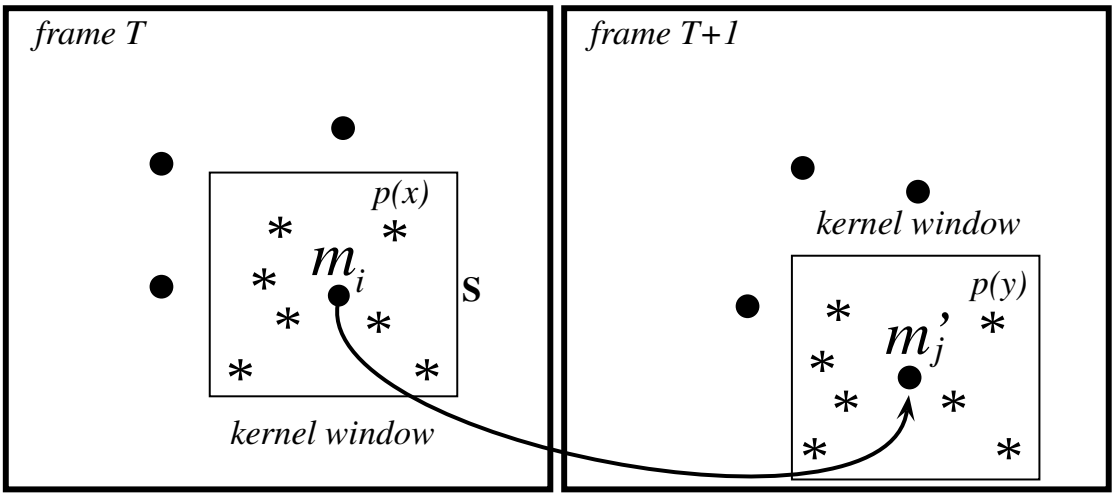

Figure 3-7: BRIEF [35] description and matching for image features (keypoints)

The feature matching used in the case studies performed in this work search for correspondent points in the current frame. Only points that are highly descriptive invariant features, called keypoints, are tested. Those keypoints were detected using

FAST [44] in Section 3.4. After the keypoints are detected they need to be described and the respective matching point should be found. Since web and handled devices have limited computational power having local descriptors that are fast to compute, to match and being memory efficient are important aspects, for that reason, was used an efficient method called Binary Robust Independent Elementary Features (BRIEF) [6].

To generate the binary string for each key-point found in the smoothed frame, the individual bits are obtained by comparing the intensities of pairs of points, $(\mathbf{p}; x, y)$, represented by $*$ symbol on Figure 3-7, along the kernel window centered on each key-point without requiring a training phase. Empirically, this technique shows that 256 or even 128 bits [6], often suffice to obtain very good matching results. The best spatial arrangement of the tested ($\mathbf{x}$, $\mathbf{y}$)-pairs of points are reach when selected based on an isotropic Gaussian distribution [6]. To compute the Gaussian distribution can be time consuming. As an optimization proposed by this article, the Gaussian distribution could be simply replaced by a random function due to its random characteristics.

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

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

where $\mathbf{p(x)}$ is the pixel intensity. The set of binary tests is defined by the n_d ($\mathbf{x}$, $\mathbf{y}$)-location pairs uniquely chosen during the initialization. The n_d-dimensional bit-string is our BRIEF descriptor for each key-point

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

In [6], $n_d = 128, 256, 512$ were used in the tests and any of those values yield good compromises between speed, storage efficiency, and recognition rate. In this article, $n_d = 128$ was used, since it presented good matching results and performance. The number of bytes required to store the descriptor can be calculated by $k = n_d/8$, proving that BRIEF is also a memory-efficient method. Detailed results can be found

in Chapter 4.

Once each keypoint is described with its binary string [6], they need to be compared with the closest matching point. Distance metric is critical to the performance of intrusion detection systems. Thus using binary strings reduces the size of the descriptor and provides an interesting data structure that is fast to operate with whose similarity can be measured by the Hamming distance which, on desktop implementations, the computation time could be driven almost to zero using the POPCNT instruction from SSE4.2 [9]. Only the latest Intel Core i7 CPUs support this instruction.

The Hamming distance is an important step on interest point matching, it provides a fast and memory-efficient way to calculate distance between binary strings. Given two image patches x and y, denote their binary descriptors as $b(x) \in \{0,1\}^n$ and $b(y) \in \{0,1\}^n$ respectively. Then their Hamming distance is computed by:

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

In which n is the dimension of binary descriptor and stands for bitwise XOR operation. According to the definition of Hamming distance, all the elements of a binary descriptor contribute equally to the distance. From the hamming distance, the Hamming weight can be calculated. It is used to find the best feature point match. Here, is generalized the Hamming distance to the weighted Hamming:

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

Where w_i is the weight of the ith element. The goal is to learn $w_i, i = 1, 2 \cdots, n$ for the binary descriptor (BRIEF) based on a set of feature points. By assigning different weights to binary codes, what we expect is to obtain a distance space in which the distances of matching patches are less than those of non-matching patches.

3.6 Homography Estimation

Typically, homographies are estimated between images by finding feature correspondences in those images. A 2D point (x, y) in an image can be represented as a 3D vector $\mathbf{x} = (x_1, x_2, x_3)$ where $x = \frac{x_1}{x_3}$ and $y = \frac{x_2}{x_3}$ [12]. This is called the homogeneous representation of a point and it lies on the projective plane P^2. A homography is an invertible mapping of points and lines on the projective plane P^2. Hartley and Zisserman [17] provide the specific definition that a homography is a mapping from $P^2 \rightarrow P^2$ is a projectivity if and only if there exists a non-singular 3×3 matrix H such that for any point in P^2 represented by vector $\mathbf{x}$ it is true that its mapped point equals $H\mathbf{x}$. It should be noted that H can be changed by multiplying by an arbitrary non-zero constant without altering the projective transformation. Thus H is considered a homogeneous matrix and only has 8 degrees of freedom even though it contains 9 elements.

The method choose to solve the homography estimation was the Direct Linear Transformation (DLT) [14, 17] algorithm. The DLT algorithm is a simple algorithm used to solve for the homography matrix H given a sufficient set of point correspondences [12].

Since we are working in homogeneous coordinates, the relationship between two corresponding points **x** and **x'** can be re-written as [12]:

$$c \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = H \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \forall \quad H = \begin{pmatrix} h1 & h2 & h3 \\ h4 & h5 & h6 \\ h7 & h8 & h9 \end{pmatrix},$$

where c is any non-zero constant, $(\ u\ v\ 1\ )^T$ represents **x'**, $(\ x\ y\ 1\ )^T$ represents **x**. Dividing the first row of equation (2.1) by the third row and the second row by the third row we get the following two equations [12]:

$$-h1x - h2y - h3 + (h7x + h8y + h9)u = 0 \tag{3.1}$$

$$-h4x - h5y - h6 + (h7x + h8y + h9)u = 0 \tag{3.2}$$

Equations (3.1) and (3.2) can be written in matrix form as $A_i\mathbf{h} = 0$. Where,

$$A_i = \begin{pmatrix} -x & -y & -1 & 0 & 0 & 0 & ux & uy & u \\ 0 & 0 & 0 & -x & -y & -1 & vx & vy & v \end{pmatrix}$$

and

$$\mathbf{h} = \begin{pmatrix} h1 & h2 & h3 & h4 & h5 & h6 & h7 & h8 & h9 \end{pmatrix}.$$

Since each point correspondence provides 2 equations, 4 correspondences are sufficient to solve for the 8 degrees of freedom of H. JavaScript typed arrays, defined in Section 2.1.4, were used in the homography estimation implementation for better performance results.

3.7 Random Sample Consensus (RANSAC)

RANSAC (Random Sample Consensus) [17] is the most commonly used robust estimation method for homographies according to [12]. The idea of the algorithm is pretty simple; For a number of iterations, a random sample of 4 correspondences is selected and a homography H is computed from those four correspondences. Each other correspondence is then classified as an inlier or outlier depending on its concurrence with H. After all of the iterations are done, the iteration that contained the largest number of inliers is selected. H can then be recomputed from all of the correspondences that were consider as inliers in that iteration [12].

One important step when applying the RANSAC algorithm described above is to decide how to classify correspondences as inliers or outliers. In the implementation for the web only assign the geometric distance [12] threshold, t, between **x'** and $H\mathbf{x}$ was enough. Hartley and Zisserman [17] provides more details about RANSAC.

Another issue is to decide how many iterations to run the algorithm, it's not required to try every combination of 4 correspondences. The goal becomes to determine the number of iterations, N, that ensures with a probability p that at least one of the random samples will be free from outliers. $N = 100$ was used on the web

implementation.

3.8 Pose Estimation

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

3.9 Rapid Object Detection (Viola Jones)

Rapid Object Detection [46] technique, much known as Viola Jones [46], brings together new algorithms and insights to construct a framework for robust and extremely rapid object detection. What has motivated this technique to be added to *tracking.js* library was the task of face detection. Surprisingly, the algorithm became robust enough to detect any training data [46], not only for faces. Currently, *tracking.js* supports, face, eyes, upper body and palm detection.

In order to scan faces, eyes or palm from images, a training phase is required. The training phase generate cascading stages. The cascade are constructed by training classifiers using AdaBoost [46] and then adjusting the threshold to minimize false negatives. OpenCV library [4] have some open-source training data, therefore doubling efforts on training is unnecessary, i.e. the face training set consisted of 4916 faces, extracted from images downloaded during a random crawl of the world wide web [46]. Those faces were scaled and aligned to a base resolution of 24 by 24 pixels [46], see Figure 3-8. Training is not the focus of this work, the algorithm to scan the faces is. The training data itself is useless if a Scanning Detector [46] is not available, the scanning is what makes the rapid object detection.

A scanning detector was implemented in JavaScript [31] and is available on *tracking.js*. The training data used is from OpenCV library [4] converted from Extensible Markup Language (XML) [5] to JavaScript Object Notation (JSON) [10]. JSON [10] has much superior performance since it's interpreted by JavaScript language [31, 10]. The results of the JavaScript [31] implementation can be used in real-time applications, the detector runs at 15 frames per second. For more information about

Figure 3-8: Example of frontal upright face images used for training [46]

performance see Chapter 4.

The overall idea of the detection process is that it uses a degenerate decision tree, what Viola and Jones [46] call "cascade". A positive result from the first classifier triggers the evaluation of a second classifier which has also been adjusted to achieve very high detection rates [46]. A positive result from the second classifier triggers a third classifier, and so on [46]. The main steps of the scanning algorithm are:

1. Create or scale a squared block, initially set to 20×20 pixels, by 1.25 per iteration;

2. Loop the squared block by Δ pixels over the image;

3. For each squared block location, loop the decision tree and evaluate each stage;

4. A positive result of the stage [46] triggers the next stage, otherwise stops the stages loop;

5. If all stages were positively evaluated store that rectangle as a possible face;

6. Once the the decision tree is done, group the overlapping rectangles;

7. Find the best rectangle of each the group to represent the face. This phase is also known as "merging phase".

The final detector is scanned across the image at multiple scales and locations of the image. This process makes sense because the features can be evaluated at any scale with the same cost [46]. Good results were obtained using a set of scales a factor of 1.25. Subsequent locations are obtained by shifting the window some number of pixels Δ, for a better accuracy $\Delta = 1$ is recommended. We can achieve a significant speedup by setting $\Delta = 2$ with only a slight decrease in accuracy, thus this value was set as default value of the JavaScript [31] implementation.

Viola and Jones [46] proposed that for each found possible rectangle representing the face to be partitioned into disjoint subsets data structures. Two detections are in the same subset if their bounding regions overlap [46]. The corners of the final bounding region are the average of the corners of all detections in the set. In order to perform well on the web, some optimizations were made in the implementation level of the scanning detector. The disjoint set was replaced by an alternative logic that is called "Minimum Neighbor Area Grouping" by this thesis. Minimum Neighbor Area Grouping has $O(N^2)$ performance [3] and consists in a loop trough the possible rectangle faces returned by the scanning detector. For each step of the loop compare the current rectangle with all other not yet compared rectangles. If the rectangle area overlaps more than η with the compared, by default $\eta = 0.5$ (or 50%), select the smallest rectangle in area of the comparison. Using the smallest rectangle, guarantees that the best match is much centralized in the face.

For more information about the JavaScript [31] implementation, such as evaluation and results, see Chapter 4.

3.10 Color Tracking Algorithm

Colors are part of our lives, they are everywhere in every single object. Being able to use colored objects to control your browser using the user camera is very appealing.

For that reason, *training.js* implemented a basic color tracking algorithm that resulted in an real-time frame rate trough a simple and intuitive API.

Chapter 4

Evaluation

4.1 Tools

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

4.2 Scenario Description

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

4.3 Evaluation Methodology

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

4.3.1 Matching Robustness

4.3.2 Oclusion Robustness

4.3.3 FPS

4.4 Results

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Chapter 5

Conclusion

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

5.1 Contributions

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

5.2 Future Work

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Bibliography

[1] Eduardo A Lundgren Melo. Typed Arrays Performance, 2013.

[2] Steve Benford, Chris Greenhalgh, Gail Reynard, Chris Brown, and Boriana Koleva. Understanding and constructing shared spaces with mixed-reality boundaries. *ACM Transactions on Computer-Human Interaction*, 5(3):185–223, 1998.

[3] Paul E Black. Big-O Notation. *Dictionary of Algorithms and Data Structures*, 2007.

[4] G Bradski. The OpenCV Library. *Dr Dobbs Journal of Software Tools*, 25(11):120–125, 2000.

[5] Tim Bray. Extensible Markup Language (XML), 2013.

[6] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF : Binary Robust Independent Elementary Features. *Computer*, 6314(3):778–792, 2010.

[7] Y Cho, J Lee, and U Neumann. A Multi-ring Color Fiducial System and an Intensity-Invariant Detection Method for Scalable Fiducial-Tracking Augmented Reality. *In IWAR*, pages 1–15, 1998.

[8] WHATWG Community. The canvas element, 2013.

[9] Intel Corporation. Intel® SSE4 Programming Reference, 2007.

[10] Douglas Crockford. JSON. 2013.

[11] Voxeo Daniel C. Burnett, Ericsson Adam Bergkvist, Cisco Cullen Jennings, and Anant Narayanan. Media Capture and Streams, 2013.

[12] Elan Dubrofsky. *Homography Estimation.* Essay, The University of Britsh Columbia, 2009.

[13] Xiph.Org Foundation. Vorbis specification, 2012.

[14] Jonas Gomes and Luiz Velho. *Fundamentos da Computação Gráfica.* 2009.

[15] Rafael C Gonzalez and Richard E Woods. *Digital Image Processing (3rd Edition).* Prentice Hall, 2007.

[16] Alan Grosskurth and M.W. Michael W Godfrey. A reference architecture for web browsers. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 661–664. IEEE, IEEE, 2005.

[17] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*, volume 2. Cambridge University Press, 2004.

[18] David Herman and Kenneth Russell. Typed Array Specification, 2013.

[19] Ian Hickson. HTML 5 Nightly Specification (W3C), 2013.

[20] Kato Hirokazu. ARToolKit: Library for Vision-based Augmented Reality. *IEIC Technical Report Institute of Electronics Information and Communication Engineers*, 101(652(PRMU2001 222-232)):79–86, 2002.

[21] Adobe Inc. Adobe Flash, 2013.

[22] Apple Inc. Safari Browser, 2013.

[23] Apple Inc. The WebKit Open Source Project, 2013.

[24] Apple Inc. WebGL Specification, 2013.

[25] Google Inc. Google Chrome Browser, 2010.

[26] Google Inc. Blink, 2013.

[27] Google Inc. HTML5 Rocks Tutorials. 2013.

[28] Google Inc. Project Glass, 2013.

[29] Google Inc., Mozilla Inc., and Opera Inc. WebRTC, 2013.

[30] Metaio Inc. Metaio Unifeye Viewer. 2009.

[31] Ecma International. ECMA-262 ECMAScript Language Specification. *JavaScript Specification*, 16(December):1–252, 2009.

[32] ISO. Information technology – Generic coding of moving pictures and associated audio information – Part 7: Advanced Audio Coding (AAC), 2006.

[33] Hirokazu Kato and Mark Billinghurst. JSARToolkit a JavaScript port of FLARToolKit, 2011.

[34] D W F Van Krevelen and R Poelman. A Survey of Augmented Reality Technologies , Applications and Limitations. *International Journal*, 9(2):1–20, 2010.

[35] Vincent Lepetit and Pascal Fua. Monocular Model-Based 3D Tracking of Rigid Objects. *Foundations and Trends® in Computer Graphics and Vision*, 1(1):1–89, 2005.

[36] Microsoft. Silverlight, 2013.

[37] Pranav Mistry, Pattie Maes, and Liyan Chang. WUW - Wear Ur World - A Wearable Gestural Interface. *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, 68(3):4111–4116, 2009.

[38] Inc. Mozilla. Gecko, 2013.

[39] Inc. Mozilla. Mozilla Developer Network, 2013.

[40] Inc. Mozilla. Mozilla Firefox Browser, 2013.

[41] João Paulo, S M Lima, Pablo C Pinheiro, Veronica Teichrieb, and Judith Kelner. Markerless Tracking Solutions for Augmented Reality on the Web.

[42] Les A Piegl. *Fundamental developments of computer-aided geometric modeling*. Academic Pr, 1993.

[43] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: a machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105–119, 2010.

[44] Edward Rosten, Reid Porter, and Tom Drummond. Machine learning for high-speed corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):1–14, 2010.

[45] Veronica Teichrieb, Monte Lima, Eduardo Lourenc, Silva Bueno, Judith Kelner, and Ismael H F Santos. A Survey of Online Monocular Markerless Augmented Reality. *International Journal of Modeling and Simulation for the Petroleum Industry*, 1(1):1–7, 2007.

[46] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518. IEEE Comput. Soc, 2001.

[47] W C. The World Wide Web Consortium (W3C), 2006.

[48] W3C. Descriptions of all CSS specifications. 2013.

[49] Wikimedia. Wikimedia Traffic Analysis Report - Browsers, 2013.

[50] Xiph.Org Foundation. Theora Specification, 2011.

[51] Yongxin Yan and Xiaolei Zhang. Research and analysis of the Virtual Reality with FLARToolKit, 2011.

[52] Feng Zhou, Henry Been-lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. *2008 7th IEEEACM International Symposium on Mixed and Augmented Reality*, 2(4):193–202, 2008.