

Video Tracking & 3D Visualization Web Application

By Eleftherios Kalykakis

Technological Educational Institute of Crete, Department of
Informatics Engineering, School of Applied Technology, 2018

THESIS PROJECT

Supervisor:

Major Professor

Athanasios G. Malamos

Technological Educational Institute of Crete

Department of Informatics Engineering

Περιεχόμενα

List of Figures.....	2
Abstract	4
1 Introduction and Motivation	5
2 Object Tracking Background.....	8
2.1 What is Object Tracking?.....	8
2.2 Tracking vs Detection	8
2.3 Object Tracking Algorithms	9
2.3.1 Multiple Instance Learning Tracker (MIL).....	9
2.3.2 Tracking Learning Detection Tracker (TLD)	11
2.3.3 Kernel Correlation Filters (KCF)	13
2.4 Tracking on Web.....	17
2.4.1 OpenCV.js	17
2.4.2 Tracking.js.....	18
3 Web Technologies & 3D Visualization.....	21
3.1 HTML 5 video.....	21
3.2 3D in web & HTML5 3D Web frameworks.....	22
3.2.1 Three.js	22
3.2.2 Babylon.js	23
3.2.3 PlayCanvas.....	25
3.2.4 X3DOM	26
3.2.5 Pros and Cons	27
3.2.6 Software Wrapper	28
4 Methodology & Implementation	28
4.1 Main Idea.....	28
4.2 Workflow & Difficulties	28
4.3 The C++ Tracking code.....	29
4.4 The App Interface	32
4.5 Sessioning	37
5 Experimental Results.....	38
5.1 Experiments Execution	38
5.2 Results Evaluation	42
6 Conclusions.....	43
6.1 Future Work	43

List of Figures

1.1 Sentioscope [1] equipment	5
1.2 Sentioscope [1] in action	6
1.3 Sportcast [2] in action.....	6
1.4 Sportcast [2] in action.....	6
1.5 Optasports [4] tracking position data.....	7
2.1 Tracking versus detection [6]	8
2.2 Pseudocode of MIL tracker [9]	9
2.3 Online MILBoost [9] pseudocode	10
2.4 Tracking by detection with a greedy approach [9].....	11
2.5 Updating a discriminative appearance model [9]	11
2.6 The block diagram of TLD [10].....	11
2.7 P-N block Diagram workflow [10].....	12
2.8 TLD Block Diagram [10]	13
2.9 Illustration of circulant matrix- The rows are cyclic shifts of a vector image [11]	13
2.10 MATLAB Algorithm with Gaussian kernel [11]	15
2.11 Table of parameters used in experiments. [11]	15
2.12 Kernelized Correlation Filter (KCF) results compared to Struck and TLD. [11]	16
2.13 Precision plot for all 50 sequences [11]	16
2.14 OpenCV logo [12]	17
2.15 Face Detection with HAAR Cascades [19]	18
2.16 Face Detection on Video Stream [19]	18
2.17 Background subtraction [19]	18
2.18 Meanshift and Camshift example [19]	18
2.19 Tracking.js Tracker Declaration and Usage [13]	18
2.20 Tracking.js video Color Tracker [13]	19
2.21 Tracking.js Color Tracker in image [13]	19
2.22 Tracking.js Object Tracker [13].....	20
3.1 Flash video source [22].....	Σφάλμα! Δεν έχει οριστεί σελιδοδείκτης.
3.2 HTML5 video source [22].....	Σφάλμα! Δεν έχει οριστεί σελιδοδείκτης.
3.3 Three.js rotating cube Example [24]	23
3.4 Simple Three.js scene setup with a rotating cube [24]	23
3.5 Babylon.js structure documentation [25]	24
3.6 Babylon.js example result [25]	24
3.7 Babylon.js example source code [25].....	25
3.8 PlayCanvas online game engine environment [26].....	25
3.9 PlayCanvas online 3D Editor [26]	26
3.10 SVG, canvas, WebGL and X3DOM relation [27]	26
3.11 X3DOM code in HTML [27].....	27
3.12 X3DOM rendered result [27].....	27
4.1 Algorithm Execution Process.....	29
4.2 Tracking Process Workflow	30
4.3 3D Transformation formulas	30
4.4 Simplified Transformation from 2D to 3D coordinations.....	31
4.5 Server algorithm workflow diagram.....	31

Video Tracking & 3D Visualization Web Application

4.6 Video Upload Page 1	32
4.7 Video Upload Page 2	32
4.8 Tracking Page.....	32
4.9 Visualization Page.....	32
4.10 Website Navigation Diagram.....	33
4.11 Upload Video workflow diagram.....	34
4.12 Tracking page Usage.....	34
4.13 Track workflow diagram	35
4.14 Visualization Page Usage	36
4.15 Visualization page workflow diagram	36
4.16 Animate function workflow diagram	37
4.17 MoveAvatar function workflow diagram	37
5.1 Position Estimation Calculator	38
5.2 Experimental Example 1.....	38
5.3 Experimental Example 2	39
5.4 Experimental Example 3	39
5.5 Experimental Example 4	40
5.6 Experimental Example 5	40
5.7 Experimental Examples Table.....	41

Abstract

The entry of technology into sports has been enormous over the last years. It has covered almost every part of all existing sports and data are being mined from everything that can affect a sport's event flow. From calculating the distance travelled from an athlete or the height that jumped or the distance threw an object, to how much effort he makes to achieve a result, sometimes a time limit, sometimes a speed limit and much more things. Some of these calculations require the usage of specific equipment either wearable or pre-installed in the environment. Although a big amount of data is exported from video analysis all these processes that are involved to this procedure. Video Tracking or Object Tracking is a major process in the hands of sports' data analysts, who use video analysis to export data, that gives them the opportunity to derive a huge amount of data only by analyzing a video. In this project an idea of a web application that helps every person interested in getting data from video analysis in a football match is implemented. This application allows user to mark and track a player's position in a video, transform it to 3D world coordinates and visualize them into a virtual 3D environment.

1 Introduction and Motivation

Over the years, during a sports event, football, basketball, Olympics etc., as spectators or participants we all wanted to have more data about what we see. To be able to see the details behind the viewable part. All these data and statistics that show how good is a player or how efficiently moves in the area. This forced people to start implementing various solutions about this. The first implementations were about keeping statistics of matches and players, a thing that helped in keeping history data of all events and achievements. Later on, with the insertion of video and replay ability things became more interesting and powerful. That was the point where companies came into the game and implemented various solutions about it. Replay analysis, and highlighted spots on videos were the first things that shown up. The last years and with the evolution of technology, those systems evolved also, and more things became possible to happen, such as 3D video replay analysis. A cameras network covers key spots of the pitch and by combining their images, there is the ability of creating an all-around optic view of the pitch. As an evolution of this system, some football associations installed cameras on the stadiums with a general optic view of the pitch, helping them spot and track players during the whole time of the match, providing this way high precision statistics about teams and players. It started being implemented in other sports, but it's not that much used as in football. This insertion is also exploited from the teams participating in the league, using these data for their own improvement. Over the years there have been a lot of video tracking applications and software to recognize and track objects inside a video. Especially in sports events where all the possible exported positioning or movement data can bring useful conclusions about the players, to help them evolve and react better and faster in similar situations. A lot of companies and organizations have come up with software and solutions about this. Some examples are **Sentioscope** an application of Sention Sports Analytics [1]. In Sentioscope players are marked in the video and get tracked and their 3D positions are provided in real-time giving the impression of a live radar/map of the field, in addition with other useful data for each player such as his covered. **Sportcast** [2] A company of the DFL German Football League [3], which is responsible for exporting useful data from video analysis and tracking for German Bundesliga. It uses a camera system that gives a general point of view of the pitch that gives the opportunity to the operators to have a complete access to all the action area of the match, the pitch. They track the players and many other statistics which help them provide as good as possible a match review in numbers.

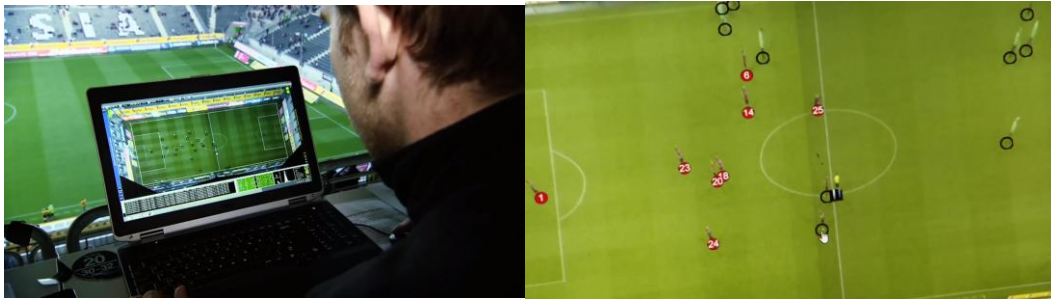


1.1 Sentioscope [1] equipment

Video Tracking & 3D Visualization Web Application



1.2 Sentioscope [1] in action

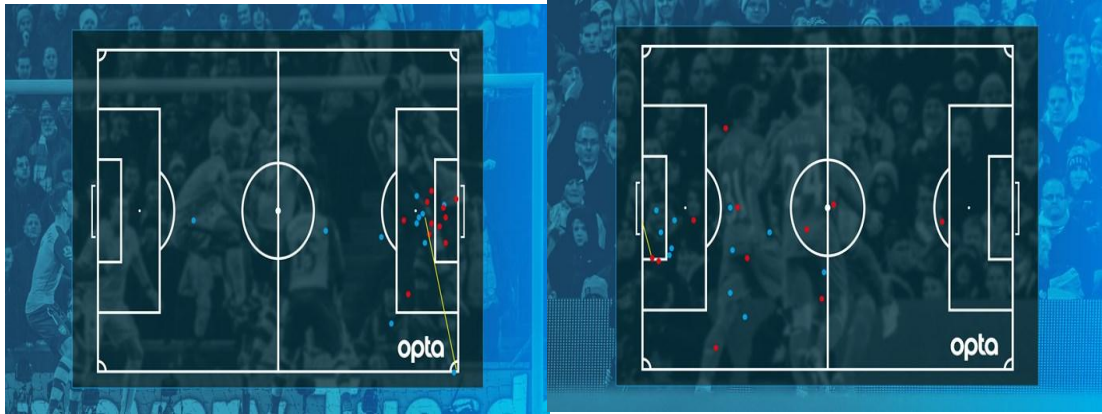


1.3 Sportcast [2] in action



1.4 Sportcast [2] in action

Another similar company example is **Opta Sports** [4], who apart from football and some of its most famous leagues coverage such as English Premier League, MLS, LaLiga etc. They are active also in other sports such as Basketball, Rugby, Baseball etc.



1.5 Optasports [4] tracking position data

Apart from professional implementations, in a more general scope apart from sports, big interest is shown about computer vision from the academic community with numerous of researchers and students are working on various object recognition, object tracking, motion tracking etc. projects, a thing that helps the greater computer vision community evolve day-by-day and provide a big amount of solutions to new people who get involved with it. Unfortunately, all these implementations either professional or not, are all stand-alone applications. There aren't any applications doing tracking and visualizing a player's position from video to 3D in web environment. The most accurate relevant example was an implementation from lusob [5], who created a football match tracker using only web technologies (HTML & JavaScript), that tracks players and ball. But it doesn't export any data and works only with specific colors. So, in that point the idea of creating a web application that can be used from everybody, professional or not, and will give him the opportunity to track player or players from whatever football video he wants and visualize their positions real-time in 3D world shows up. Based on the instant usage of web and mobile devices, so the user can extract data from video by using only his phone or a web browser. The application is created strictly with web technologies and works on web browser environment. It enables user to:

- Upload a video
- Mark the players he wants and track them
- Export and visualize their positions in 3D-world coordination

The application will be able to run anywhere, because its built primarily with web technologies that are compatible with the most existing active browsers, which gives us the advantage of mobility. The tracking algorithm is the only part that it's not built with web technologies mainly because of the demanding on resources to calculate positions in each frame. So, it built it using C++ in combination with OpenCV library. It's a frame-by-frame tracking process implementing MIL tracking algorithm, with a position transform formula included from 2D to 3D for each object's position detected on screen. With this transformation the results are exported ready on 3D world coordinates, something that does the visualization process easier.

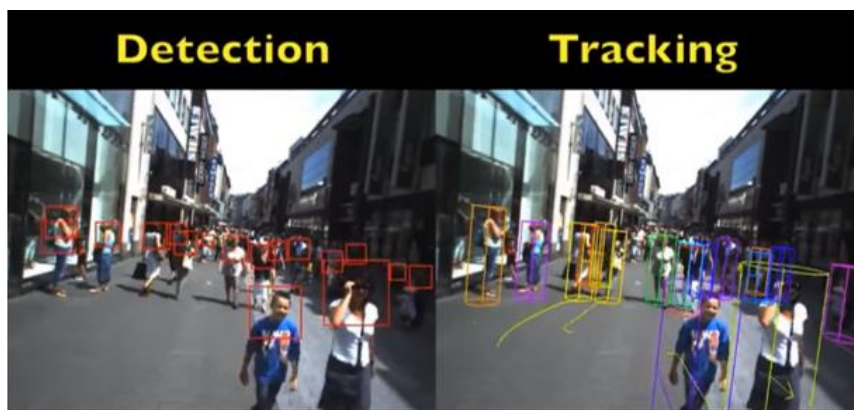
2 Object Tracking Background

2.1 What is Object Tracking?

As Object Tracking we call the process of locating and following one or more moving objects into a video's frame sequence [6]. It's a process that is commonly used in computer vision and machine learning and sometimes it's related to these terms. There are two types of trackers for simple and multiple tracking processes. In simple trackers the tracked object's position on frame is specified with a rectangle and according to the tracking algorithm used it "follows" the object's position in the subsequence frames. In multiple trackers we do the same position specification as in simple, but before use the tracking algorithm, an algorithm to specify the rectangles matching between frames is needed. A very well-known respective algorithm and commonly used is Kalman filtering [7], which is one of the most famous algorithms for object's location variation. There are also, Dense optical flow algorithms, who estimate the alteration of every pixel in a video frame and Sparse algorithms, who track the location of feature points in an image. Kanade-Lucas-Tomasi (KLT) [8], is a typical example of Sparse optical flow algorithm.

2.2 Tracking vs Detection

But here arises a serious question. Why tracking and not just repeated detections? The answer is clear. Detection algorithms are slower than tracking algorithms. When a detected object in the previous frame is tracked, many information about object's appearance are obtained.



2.1 Tracking versus detection [6]

Location and speed of its motion and the direction in the previous frame, are some of them. In the currently examined frame, all this information can be used to predict the object's new position in the next frame. To locate the object more accurately, a limited search needs to be done around the expected location. A detection algorithm starts from the beginning, while a reliable tracking algorithm takes advantage of the acquired information about the object to that point. Accordingly, to design an efficient system, most of the time, an object detection is performed on every n^{th} frame, while we run a tracking algorithm in the $n-1$ frames in between. Someone may wonder, if it is more achievable to simply detect the object in the first frame and track afterwards. Tracking can take advantage of the information that are obtained, but the accuracy of the procedure is uncertain when the moving object goes behind other objects or an obstacle for a long period of time or moves too fast. In addition, tracking algorithms often accumulate errors and the bounding box that tracks the object slowly slides away from the tracked object. To deal with these problems, a detection algorithm runs from time to time. Since detection algorithms are trained on object's instances, more information is known about object's general class. In a

different manner, tracking algorithms have more knowledge about the specific instance of the class they are tracking. An efficient tracking algorithm will be able to handle occlusion, while detection fails. Further-more, tracking algorithms help preserving object's identity. An array of rectangles that contain the object, is the output of object detection, yet there is no identity attached to the object. For example, if we want to detect 10 moving circles in a video the output will be the rectangles matching to all the circles the detector has detected in one frame. In the next frame, another array of rectangles will be the output. The problem is that in the first frame, a specific circle might be defined by the rectangle at location 12 in the array and in the next frame, it could be at location 15. If detection is being used on one frame, there is no indication which rectangle corresponds to which object, and the solution is tracking, because provides a way to associate the circles.

2.3 Object Tracking Algorithms

2.3.1 Multiple Instance Learning Tracker (MIL)

Multiple Instance Learning (MIL) [9] algorithm, is developed based on the hypothesis of tracking an object in a video, knowing only its position on the first frame, and anything more about it. A solution to this problem comes with an approach of tracking by detection techniques. In this case algorithm separates object from background and subtracts the second one. MIL tracking system generates several potential positions (mentioned as positive examples) by searching in a neighbor area of present location. In MIL, positive examples are presented as sets known as “bags” which are labeled with necessary tags. It's not necessary in a positive bag all examples to be positive examples. If a positive label is attached to a bag, that means that this bag contains at least one positive indication, otherwise the bag is negative. In the authors approach of paper “*Visual Tracking with Online Multiple Instance Learning*” [9], MIL framework is selected, because it gives them the opportunity to change the appearance model by combining other images, although the image patch that accurately captures the object they are interested in, is not known. As a result, an easy to implement and effective algorithm, with fewer tweaked parameters built.

Algorithm 1 MILTrack

Input: New video frame number k

- 1: Crop out a set of image patches, $X^s = \{x | s > ||l(x) - l_{t-1}^*||\}$ and compute feature vectors.
 - 2: Use MIL classifier to estimate $p(y = 1|x)$ for $x \in X^s$.
 - 3: Update tracker location $l_t^* = l(\arg\max_{x \in X^s} p(y|x))$
 - 4: Crop out two sets of image patches $X^r = \{x | r > ||l(x) - l_t^*||\}$ and $X^{r,\beta} = \{x | \beta > ||l(x) - l_t^*|| > r\}$.
 - 5: Update MIL appearance model with one positive bag X^r and $|X^{r,\beta}|$ negative bags, each containing a single image patch from the set $X^{r,\beta}$
-

2.2 Pseudocode of MIL tracker [9]

This process trains an online classifier to have more secure and accurate results and deal with the situation of drifting from wrong labeling of the samples by the tracker. MILBoost [9] is the algorithm on which paper's [9] authors used to produce their MILTrack [9] system.

Algorithm 2 Online-MILBoost (OMB)

Input: Dataset $\{X_i, y_i\}_{i=1}^N$, where $X_i = \{x_{i1}, x_{i2}, \dots\}$, $y_i \in \{0, 1\}$

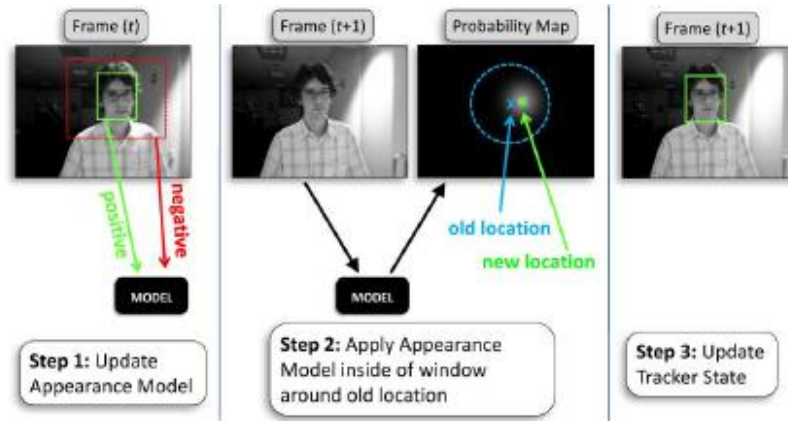
- 1: Update all M weak classifiers in the pool with data $\{x_{ij}, y_i\}$
- 2: Initialize $H_{ij} = 0$ for all i, j
- 3: **for** $k = 1$ to K **do**
- 4: **for** $m = 1$ to M **do**
- 5: $p_{ij}^m = \sigma(H_{ij} + h_m(x_{ij}))$
- 6: $p_i^m = 1 - \prod_j (1 - p_{ij}^m)$
- 7: $\mathcal{L}^m = \sum_i (y_i \log(p_i^m) + (1 - y_i) \log(1 - p_i^m))$
- 8: **end for**
- 9: $m^* = \operatorname{argmin}_m \mathcal{L}^m$
- 10: $\mathbf{h}_k(x) \leftarrow h_{m^*}(x)$
- 11: $H_{ij} = H_{ij} + \mathbf{h}_k(x)$
- 12: **end for**

Output: Classifier $\mathbf{H}(x) = \sum_k \mathbf{h}_k(x)$, where $p(y|x) = \sigma(\mathbf{H}(x))$

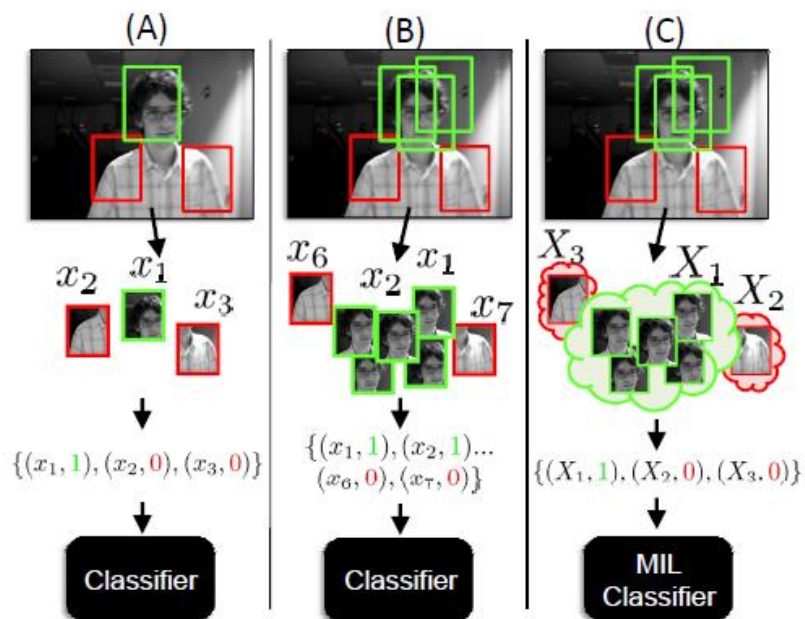
2.3 Online MILBoost [9] pseudocode

So how their [9] approach work. A discriminative classifier is composing the appearance model, that estimates position based on the image's patch representation in feature space. The object's existence in the image of interest is represented with a variable. The tracker maintains the object location at every time step, to achieve better computational efficiency and simplicity in the algorithm. Then a set of images, is cropped out of each new frame, if they exist in the search area of the tracker. The radius delimits the definition of positive instances during initialization. In the tracker's location update process a greedy strategy is used to avoid maintaining a new location for each frame. Alternatively, a model based in motions choses, tracker is expected to appear at every step time, of course within the specified radius. After the update of the tracker's location, the appearance model is updated also, using the revised version of MIL algorithm which is now trained with labeled bags. So, as soon as the set of patches is cropped out, this bag is positive labeled when their distance from center point is smaller than radius.

Video Tracking & 3D Visualization Web Application



2.4 Tracking by detection with a greedy approach [9]



2.5 Updating a discriminative appearance model [9]

2.3.2 Tracking Learning Detection Tracker (TLD)

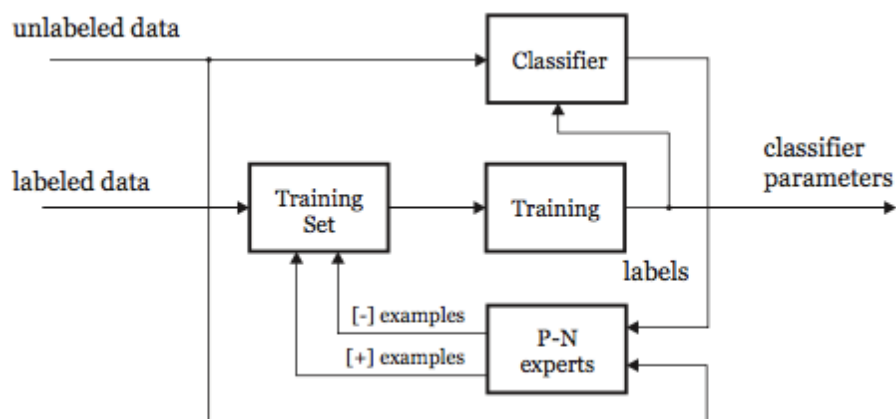
TLD (Tracking, learning and detection) [10] algorithm breaks down the process of long-time learning, in three sub tasks: Tracking, Learning, Detection



2.6 The block diagram of TLD [10]

The object is tracked in every frame. During Detection procedure the detector calculates the frequency of appearances of the object and corrects the tracker. The Learning process keeps track of the errors happened in order not to happen again. Tracking procedure collects the revised data from previous two and tracks the object. This algorithm introduces P-N learning method. Its main process is to evaluate the detector in each video frame. As mentioned, the responses returned are processed by two types of maven, P that undermines the missing detections and N that undermines the wrong results. The detector is constantly trained to avoid same errors happen again. In a case that the object is visible and the motion from frame-by-frame is limited, the Tracker approximates the motion of the model in successive frames. If the object leaves the camera view, will cause failure of the Tracker and won't have any chance to recover. The Detector does a full examination of the image in each frame independently, to localize the whole range of appearances that have been learned and observed in the past. The errors types of this Detector are as common false negatives and false positives. The performance of the Tracker and Detector is observed by the Learning process, which is responsible for generating training examples, to help the Detector avoid errors in future processes. The P-N learning is expressed by the following stages:

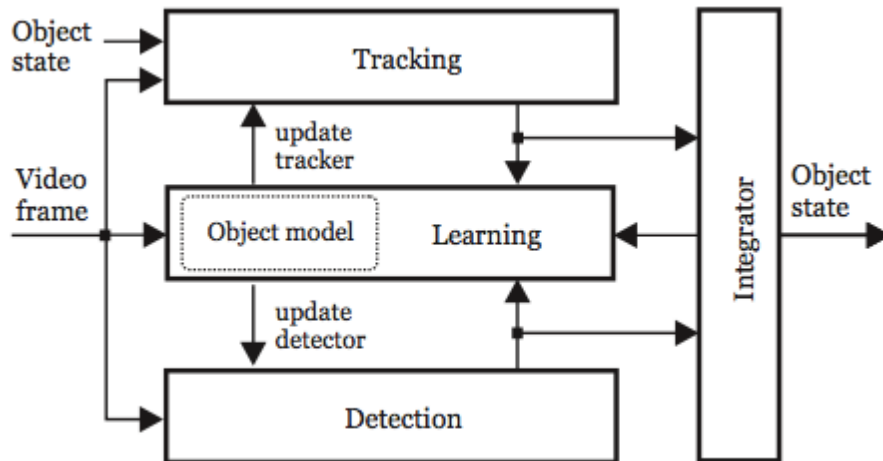
- a training classifier
- a set of labeled bags of instances, that form the training set
- the training method for the classifier
- functions named P-N, generated during learning negative and positive instances.



2.7 P-N block Diagram workflow [10]

To initialize the training process, a labelled set is imported to the set for train. Then, this set is passed to the classifier that gets trained. The estimation of the classifiers error is a critical point in P-N method. A false-positives estimation must be separated from false-negatives estimation. Based on the current classification, the P-expert splits the unlabeled set in two parts and each part is analyzed. Examples classified as negatives, are analyzed and P-expert estimates false-negatives, labels them as positive and adds them to the training set. Examples classified as positives, are analyzed by N-expert, which estimates false-positives, labels them as negative and adds them also to the training set. The purpose of P-experts is increasing the generality of the classifier by discovering new appearances of the object. Generating positive training examples, is achievable if the P-expert identities are parts of trajectory, which are reliable. The N-experts increases the classifiers discriminability, by the

assumption that the object takes over one location in the image. The locations surrounding is marked as negative when the location of the object is known.

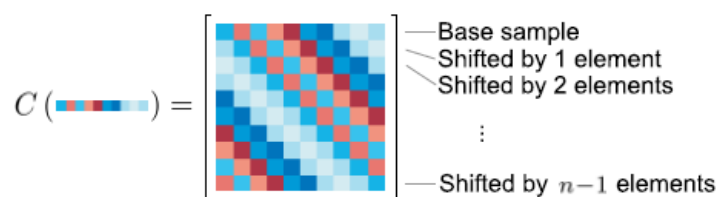


2.8 TLD Block Diagram [10]

2.3.3 Kernel Correlation Filters (KCF)

A discriminative classifier that separates the surrounding environment from the target object, is considered as basis for most state-of-the-art trackers. A natural image can change through time, so this classifier needs to be trained with sample patches, which are scaled and translated. A model of data sets of moved patches was proposed from « *High-Speed Tracking with Kernelized Correlation Filters* » paper [11]. According to Fourier, if a translation model is used, learning algorithms turn out to be easier as more samples are added. Correlation filters benefit from the fact that in Fourier, the convolution of two patches corresponds to an element-wise product. Thus, correlation filters can indicate linear classifiers output for image shifts or several translations straightaway. Authors [11] focused on Ridge Regression with classical correlation filters and cyclically sifted samples, that enables fast learning with $O(n \log n)$ Fast Fourier Transforms. The main objective of the training process is to find a function which will minimize the possibility of error. The basic sample is a vector representing the object of interest, noted as x . The classifier needs to be trained with several virtual samples and the base sample, which must be translated. The cyclic shift operator models' one-dimensional translations of this vector.

$$P = \begin{bmatrix} 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$



2.9 Illustration of circulant matrix- The rows are cyclic shifts of a vector image [11]

The functionality of the KCF is presented in the figure below as MATLAB code

Algorithm 1: Matlab code, with a Gaussian kernel.
 Multiple channels (third dimension of image patches) are supported. It is possible to further reduce the number of FFT calls. Implementation with GUI available at: <http://www.isr.uc.pt/~henriques/>

Inputs

- x : training image patch, $m \times n \times c$
- y : regression target, Gaussian-shaped, $m \times n$
- z : test image patch, $m \times n \times c$

Output

- responses: detection score for each location, $m \times n$

```

function alphaf = train(x, y, sigma, lambda)
    k = kernel_correlation(x, x, sigma);
    alphaf = fft2(y) ./ (fft2(k) + lambda);
end

function responses = detect(alphaf, x, z, sigma)
    k = kernel_correlation(z, x, sigma);
    responses = real(ifft2(alphaf .* fft2(k)));
end

function k = kernel_correlation(x1, x2, sigma)
    c = ifft2(sum(conj(fft2(x1)) .* fft2(x2), 3));
    d = x1(:)'*x1(:) + x2(:)'*x2(:) - 2 * c;
    k = exp(-1 / sigma^2 * abs(d) / numel(d));
end
  
```

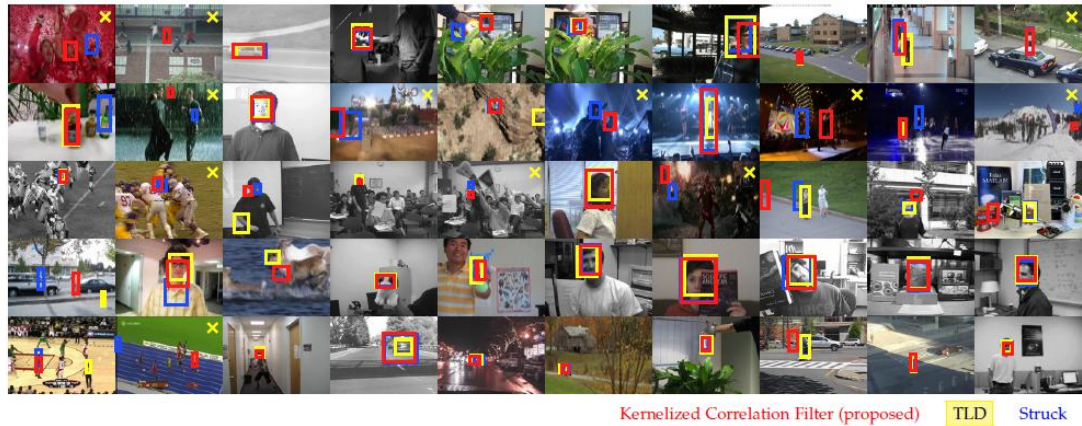
2.10 MATLAB Algorithm with Gaussian kernel [11]

In the initial frame, authors [11] train the model at the initial frame's position. To make some context available, this image patch is bigger than targets. The detection over the patch at the precedent position is done, for each new frame. This is when the targets location updates, to this that generated the higher value. Finally, at the new position, a new model is trained and "linearly interpolate the obtained values of α and x with the ones from the previous frame, to provide the tracker with some memory" [11] as authors mention. The proposed KCF approach, using Gaussian kernel is implemented in MATLAB. The algorithm is tested with two alternatives: one that takes the raw pixel values, and a second one that uses descriptors with a size of cell of 4 pixels, Felzenszwalbs variant. Parameters required by the KCF tracker, same to all videos and depicted in the following table.

KCF/DCF parameters	With raw pixels	With HOG
Feature bandwidth σ	0.2	0.5
Adaptation rate	0.075	0.02
Spatial bandwidth s	$\sqrt{mn}/10$	
Regularization λ	10^{-4}	

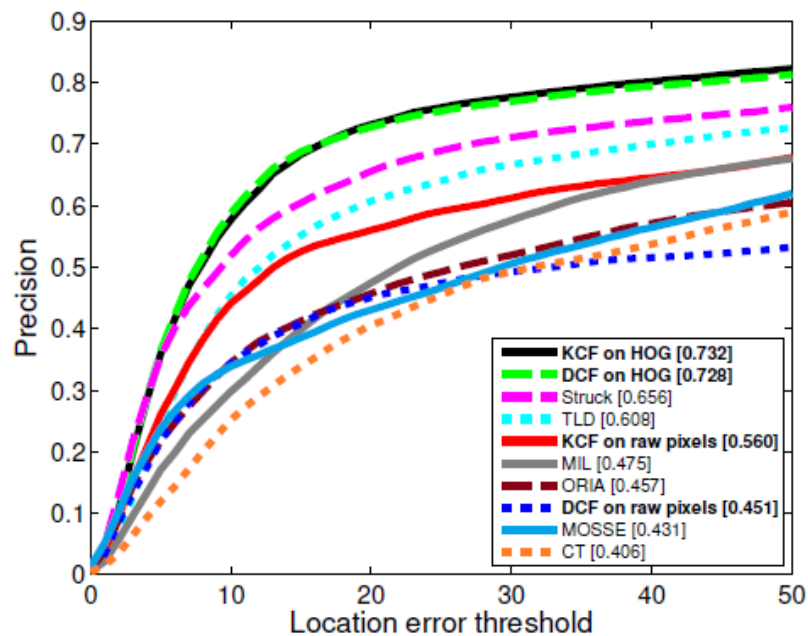
2.11 Table of parameters used in experiments. [11]

Authors [11] compared their tracking approach with TLD and Struck tracker on fifty videos dataset instead of the original tests with 12 videos, using precision curve for the performance criteria. Moreover, instead of using raw pixels, they add to the KCF tracker a new variant founded on Histogram of Oriented Gradients features. If the center of the predicted target is inside of ground truths distance threshold, then we consider that the frame is properly tracked. Precision curves demonstrate the percentage value of correctly tracked frames from a range of thresholds.



2.12 Kernelized Correlation Filter (KCF) results compared to Struck and TLD. [11]

Without any feature extraction, the KCF accomplished better results than a linear filter. With Histogram of Oriented Gradients features, nonlinear KCF surpassed TLD and Struck tracker. Tracker's speed is immediately related to the size of the tracking area and is a significant consideration when the comparison, between trackers based on relational filter, is made. They decided to reduce their tracked region, to speed feature computation, but this decision slowed down the performance of the tracker.



2.13 Precision plot for all 50 sequences [11]

Furthermore, they experimented with sequence attributes, such as occlusion and illumination changes, background clutter and out of view target. In occlusions and non-rigid

deformations, KCF with HOG features surpassed the other algorithms. TLD's performance is better in out of view sequences, because KCF lacks a failure recovery mechanism. Moreover, this algorithm performed better in background clutter, while almost all of trackers are severely affected, although it does not recover though from full occlusion.

2.4 Tracking on Web

All these algorithms generally work on standalone applications. Usually they are inherited in software solutions that work independently as programs. The most common programming languages used to write those programs are C++, Python and Java. This means that they are not able to run on a web browser environment directly, because they won't be understandable due to the absence of JavaScript, the browser's main language. Their integration effort on browser started only a few years ago and there are not that many implementations doing the same thing as standalone applications. Only OpenCV [12] and Tracking.js [13] have some ready to use implementations but there are facing more the detection part and face recognition. Tracking is used only on face and eye movement and generally over web camera.

2.4.1 OpenCV.js

OpenCV [12] is a real-time computer vision library of programming functions. Initially developed by Intel [14], supported by Willow Garage [15] later and then from Itseez which lately came by Intel. The library comes under a license which is free to use for both academic and commercial use. It comes with the most common object-oriented interfaces (C++, Java, Python) and supported from all platforms (Windows, Linux, Mac OS, iOS and Android). It was designed for efficient computations in real-time applications. Written in C/C++ gives library the advantage of multi-core processing. Also, the fact that its enabled with OpenCL, gives the ability of the hardware acceleration of the underlying platforms.



2.14 OpenCV logo [12]

After becoming worldwide known, OpenCV has a user community counting more than 47 thousand people and over 14 million downloads. It's used from art to advanced robotics. It's currently running on version 3.4.1 in all platforms. Lately, and more specific after version 3.0 release, they introduced OpenCV.js, an implementation of various library's functions in JavaScript. The purpose of the new library is to help OpenCV inheritance in web development and give the opportunity to developers and computer vision researchers, access a variety of web-based OpenCV examples. OpenCV.js uses Emscripten [16] toolchain to compile OpenCV functions into asm.js [17] or WebAssembly [18], and provide a JavaScript API to be easier accessible for web applications. According to OpenCV.js instructions page [19], the library needs to be built from default OpenCV libraries using Emscripten [16]. After build is complete a JavaScript file is produced and can be faced as a typical JavaScript file in web development. It's inserted into HTML code as all the rest files and gives the developer access to all the implemented functions of OpenCV. Below there are several examples of some functions of OpenCV.js in use:



2.15 Face Detection with HAAR Cascades [19]



2.16 Face Detection on Video Stream [19]



2.17 Background subtraction [19]



2.18 Meanshift and Camshift example [19]

2.4.2 Tracking.js

Tracking.js [13] is a computer vision framework, containing algorithm and techniques written completely in HTML5 and JavaScript. It provides the user functions for face detection, color tracking and feature detection. It has a very lightweight source code which is very important on web development. It provides a variety of Tracker types, such as color tracker, object tracker and custom tracker. They have very simple usage with just initialization as variable and track purpose task initialization needed. Then, can be used as native JavaScript objects.

```
//declaration
var myTracker = new tracking.Tracker('target');

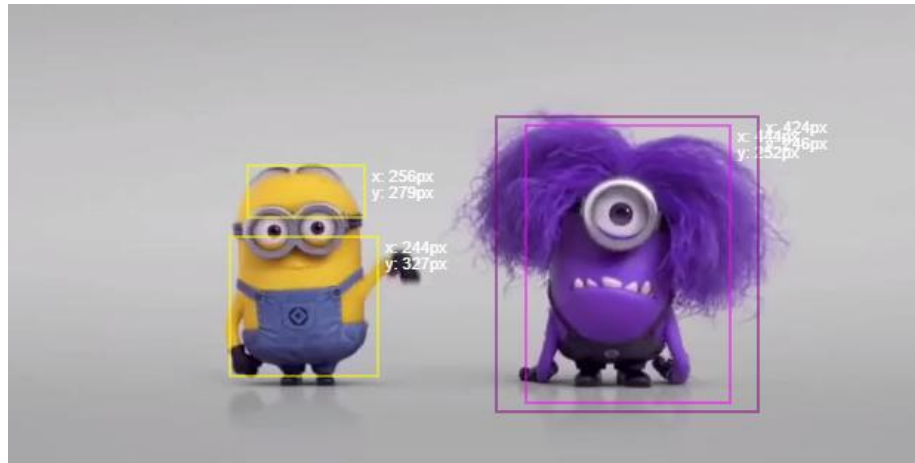
//define
myTracker.on('track', function(event) {
  if (event.data.length === 0) {
    // No targets were detected in this frame.
  } else {
    event.data.forEach(function(data) {
      // Plots the detected targets here.
    });
  }
});

//call
var trackerTask = tracking.track('#myVideo', myTracker);
```

2.19 Tracking.js Tracker Declaration and Usage [13]

Color Tracker can detect and track if we have a video the given color from the user or any of the default known colors of the framework, yellow, cyan, magenta. It can also detect any color user gives either named or hex code.

Video Tracking & 3D Visualization Web Application

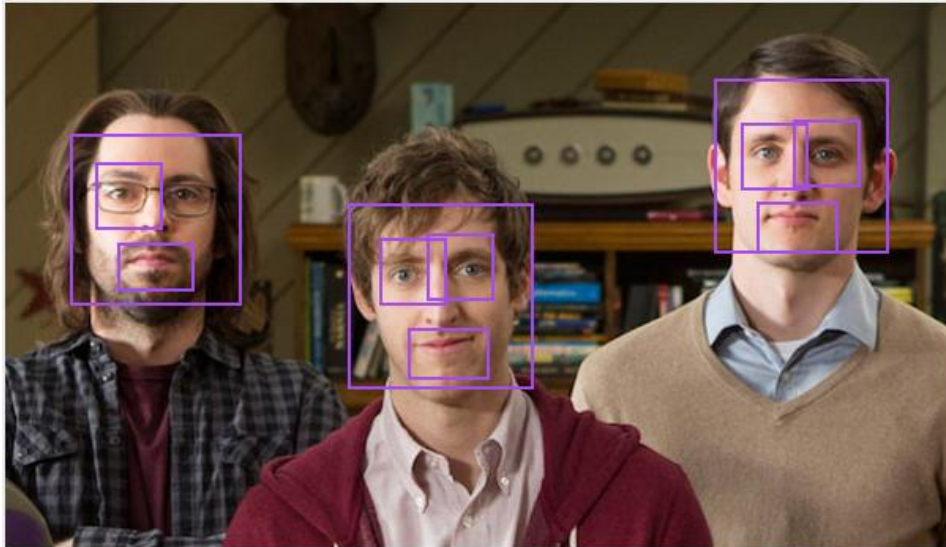


2.20 Tracking.js video Color Tracker [13]



2.21 Tracking.js Color Tracker in image [13]

Object Tracker on the other hand can recognize only prefixed classifiers. These classifiers are for face, mouth and eyes and are included inside frameworks library. User doesn't have the ability to add his own classifiers in the library. But the tracker has the same easy usage as Color Tracker with initialization and calls from the JavaScript code.



2.22 Tracking.js Object Tracker [13]

The last tracker category of the framework is Custom Tracker. As its name says is a tracker that can be created from user and track specific things user wants, for example get access to the camera and display the pixel matrix on canvas for each frame. It also has the same simple usage, but it also needs to be inherited to the Tracker hyperclass.

3 Web Technologies & 3D Visualization

3.1 HTML 5 video

HTML 5 [20] is the latest version of the HTML standard markup language and constitutes the official language of presenting data on the Web. It was first published in 2014, to improve the already existing language version with support of the latest multimedia, keeping it also both readable by humans and understood by computers and devices. HTML5 includes detailed processing models to encourage more functional implementations. For that reason, many new syntactic features are included. Elements like <video>, <audio> and <canvas> were added to handle multimedia and graphical content, in addition with SVG (scalable vector graphics) content and mathematical formulas from MathML. The <video> element [21] first presented in 2007. The developer company released a proclamation and states video as a first-rate component of the web. HTML5 video support is evolving rapidly [22]. While Flash video, the predecessor of HTML5 video, is non-searchable and often falls into incompatibilities since users usually don't have a lately updated browser, HTML5 video is more transcendent and appears to give a solution to these problems. That's because it's highly searchable, it renders efficiently on mobile devices and modern browsers, and is easy to style and integrate. All modern browsers now support HTML5 video. If we talk on statistics, over half of users have a modern browser that supports HTML5 video. To render HTML5 properly on older version browsers, developers use the video element, but they keep Flash as a backup, in case HTML5 don't work. That's achieved by embedding video in at least two of the three supported formats .mp4, .ogg/.ogv or .webm. The element also supports multiple sources. Using any number of <source> tags, the browser will choose automatically which of them to load. The ideal HTML5 video format would:

- Have good compression
- good image quality
- low decode processor use
- Be license-free
- hardware video decoder should exist for the format

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-
444553540000" width="425" height="344"
codebase="http://download.macromedia.com/pub/shockwave/
cabs/flash/swflash.cab#version=6,0,40,0">
<param name="allowFullScreen" value="true" />
<param name="allowscriptaccess" value="always" />
<param name="src"
value="http://www.youtube.com/v/oHg5SJYRHA0&hl=en&fs=1&
" />
<param name="allowfullscreen" value="true" />
<embed type="application/x-shockwave-flash" width="425"
height="344"
src="http://www.youtube.com/v/oHg5SJYRHA0&hl=en&fs=1&"
allowscriptaccess="always" allowfullscreen="true">
</embed>
</object>
```

3.1 Flash video source [22]

Video Tracking & 3D Visualization Web Application

```
<video width="640" height="360"  
src="http://www.youtube.com/demo/google_main.mp4"  
controls autobuffer>  
<p> Try this page in Safari 4! Or you can <a  
href="http://www.youtube.com/demo/google_main.mp4">down  
load the video</a> instead.</p>  
</video>
```

3.2 HTML5 video source [22]

HTML5 <video> tag comes up with a number of identities that give the video a sort of functionality. Some of them are shown below:

- Loop – lets the video continuously play
- Autoplay – the video will play once the page is opened
- Poster – indicates the images that will be shown when a video is loading
- Controls – standard play, pause, sound controls
- Preload – download video in background before start to play.

HTML5 video is also fully style customizable using CSS and CSS3. Almost all attributes can be edited. HTML5 makes video usage very easy and fast, thing that saves time to the developer and offers the client a better and easier solution.

3.2 3D in web & HTML5 3D Web frameworks

The web has always been an information transfer and visualization mean, a restricted one though, especially in the visualization part. Until lately, HTML developers either should use CSS and JavaScript for animations and visual effects for their websites or had to install plugins like Flash. With the addition of the canvas element to the language, in addition with Web GL, and SVG images, this is no longer a problem! There are so many new features that face various situations with graphics on the web, 2D and 3D. Of course, none of those additions would be efficient if they couldn't run fast. Thankfully, JavaScript has become fast enough to be able to handle 3D games and manipulating video in real-time. Also, with the implementation of Hardware accelerated compositing in browsers, CSS transitions and transforms are piece of cake. This was the motivation for developers to proceed in the creation of many web frameworks for 3D graphics, to enrich browsers visualization ability even in 3D.

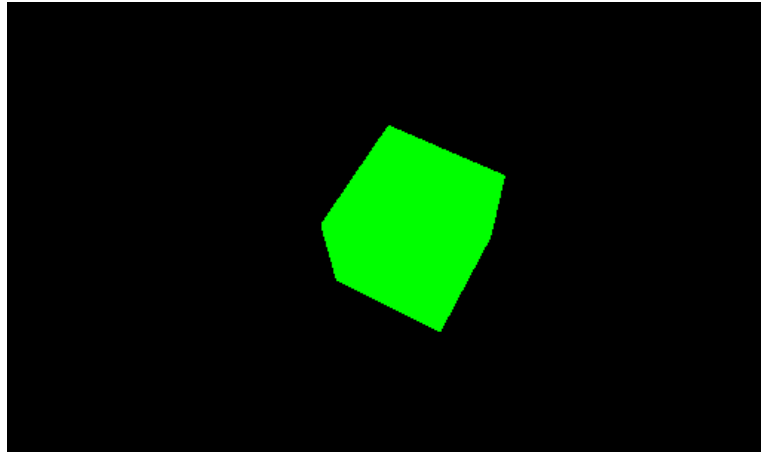
3.2.1 Three.js

Three.js is a JavaScript functions library creating and displaying 3D computer graphics in a web browser. It was first developed in 2010 under the MIT license and enforces the creation of GPU-accelerated 3D models and animations, using the JavaScript language, as components of a website without any browser plugin needed. This is achieved thanks to the WebGL library that Three.js is built on. The source code was first written in ActionScript, until 2009 when it moved into JavaScript. The two strong points of this transfer were:

- No need of pre-compile
- Cross platform support

With the addition of WebGL, the renderer became easy part to implement as Three.js was designed using rendering as module and not as core. This allows Three.js to run in every browser that supports WebGL 1.0. It includes features like [23]: Effects, Scenes, Cameras, Animation, Lights, Materials, Virtual reality etc.

[22]



3.1 Three.js rotating cube Example [24]

```
<html>
  <head>
    <title>My first three.js app</title>
    <style>
      body { margin: 0; }
      canvas { width: 100%; height: 100% }
    </style>
  </head>
  <body>
    <script src="js/three.js"></script>
    <script>
      var scene = new THREE.Scene();
      var camera = new THREE.PerspectiveCamera( 75, window.innerWidth/window.innerHeight, 0.1, 1000 );

      var renderer = new THREE.WebGLRenderer();
      renderer.setSize( window.innerWidth, window.innerHeight );
      document.body.appendChild( renderer.domElement );

      var geometry = new THREE.BoxGeometry( 1, 1, 1 );
      var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
      var cube = new THREE.Mesh( geometry, material );
      scene.add( cube );

      camera.position.z = 5;

      var animate = function () {
        requestAnimationFrame( animate );

        cube.rotation.x += 0.1;
        cube.rotation.y += 0.1;

        renderer.render( scene, camera );
      };

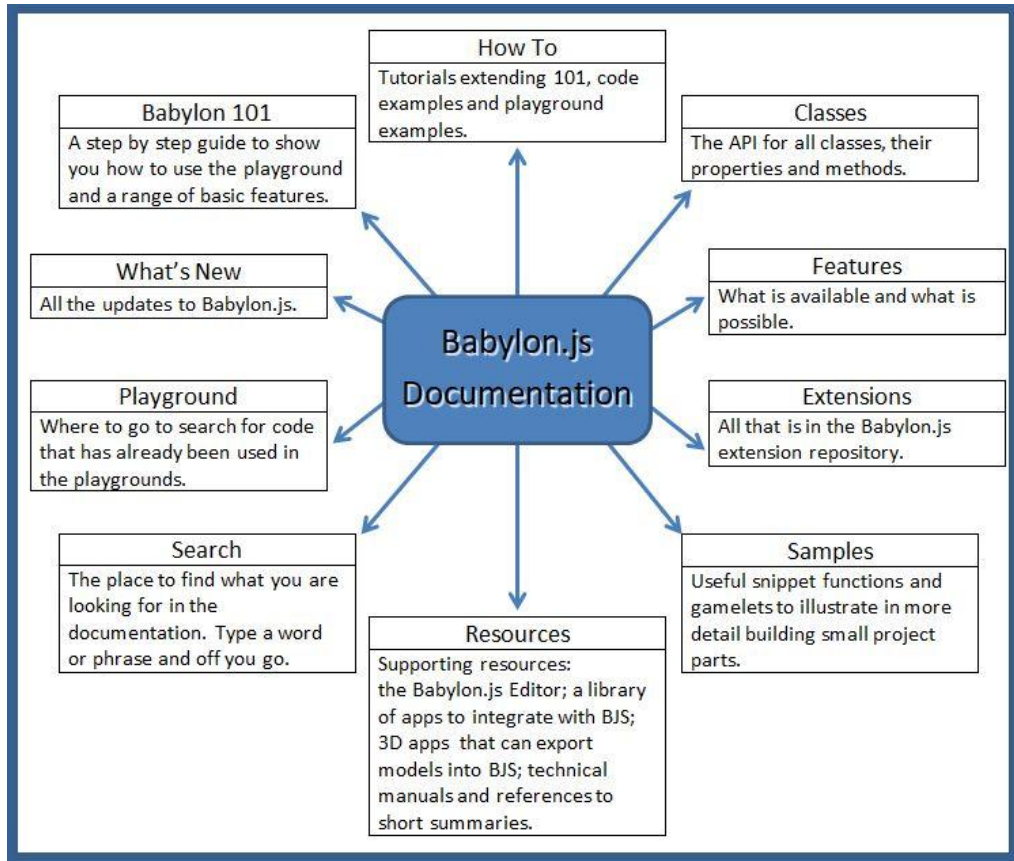
      animate();
    </script>
  </body>
</html>
```

3.2 Simple Three.js scene setup with a rotating cube [24]

3.2.2 Babylon.js

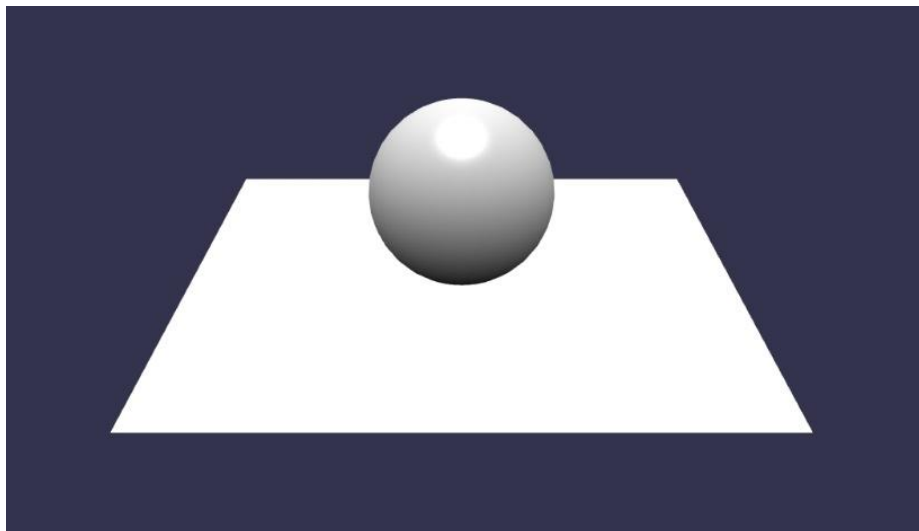
Babylon.js is an open-source WebGL/JavaScript 3DEngine supported by all modern browsers. There is not much more theoretical coverage about the framework, but it constitutes one of the most commonly used HTML5 3D web frameworks.

Video Tracking & 3D Visualization Web Application



3.3 Babylon.js structure documentation [25]

The basic example with a sphere on a plane is implemented below with the source code following the result.



3.4 Babylon.js example result [25]

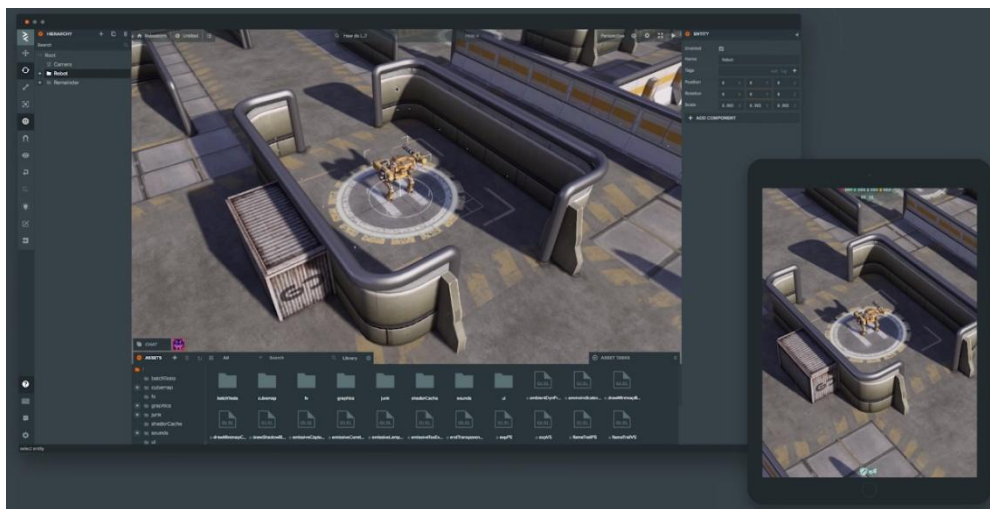
Video Tracking & 3D Visualization Web Application

```
<canvas id="renderCanvas"></canvas>
<script>
  window.addEventListener('DOMContentLoaded', function(){
    // get the canvas DOM element
    var canvas = document.getElementById('renderCanvas');
    // load the 3D engine
    var engine = new BABYLON.Engine(canvas, true);
    // createScene function that creates and return the scene
    var createScene = function(){
      // create a basic BJS Scene object
      var scene = new BABYLON.Scene(engine);
      // create a FreeCamera, and set its position to {x:0, y:5, z:-10}
      var camera = new BABYLON.FreeCamera('camera1', new BABYLON.Vector3(0, 5, -10),
scene);
      // target the camera to scene origin
      camera.setTarget(BABYLON.Vector3.Zero());
      // attach the camera to the canvas
      camera.attachControl(canvas, false);
      // create a basic light, aiming 0,1,0 - meaning, to the sky
      var light = new BABYLON.HemisphericLight('light1', new BABYLON.Vector3(0,1,0),
scene);
      // create a built-in "sphere" shape; its constructor takes 6 params: name, segment,
diameter, scene, updatable, sideOrientation
      var sphere = BABYLON.Mesh.CreateSphere('sphere1', 16, 2, scene);
      // move the sphere upward 1/2 of its height
      sphere.position.y = 1;
      // create a built-in "ground" shape;
      var ground = BABYLON.Mesh.CreateGround('ground1', 6, 6, 2, scene);
      // return the created scene
      return scene;
    }
    // call the createScene function
    var scene = createScene();
    // run the render loop
    engine.runRenderLoop(function(){
      scene.render();
    });
    // the canvas/window resize event handler
    window.addEventListener('resize', function(){
      engine.resize();
    });
  });
</script>
```

3.5 Babylon.js example source code [25]

3.2.3 PlayCanvas

PlayCanvas is going away of a simple HTML5 3D framework and approaches the concept of game engine. As they contend in their own site [24] *"The Web-First Game Engine Collaboratively build stunning HTML5 visualizations and games"*.



3.6 PlayCanvas online game engine environment [26]

[25]

Eleftherios Kalykakis © TEI Crete 2018

Video Tracking & 3D Visualization Web Application

It is open sourced under MIT license. The PlayCanvas Engine is considered as the world's most advanced WebGL game engine. It uses JavaScript to program anything from simple 2D games to advanced 3D graphics, all written in cross-platform HTML5 for every major browser and device. The major features of PlayCanvas are:

- Tiny engine footprint, which means that it loads and executes quickly
- Mobile Optimized
- Very small compile step (close to 0)
- Easy debugging and profiling

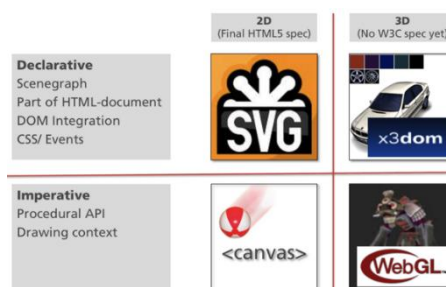
These are only the key features of the game engine. There are a lot more are mentioned on their homepage [24]. The most important tool of PlayCanvas is the online editor. It is the most advanced WebGL authoring environment available. It contains a full set of features to help speed up development giving the opportunity of real-time 3D models design from the browser and include them directly in the project.



3.7 PlayCanvas online 3D Editor [26]

3.2.4 X3DOM

X3DOM [25] is an open-source JavaScript 3D framework, used to represent 3D content in webpages. Since its developed on standard browser technology, there is no need of plugin to work properly. In a few words, with X3DOM a 3D scene can be created and displayed by using textual representation instead code. Nowadays, 3D content becomes a first-class component inside HTML, just like the other core components of the language.



3.8 SVG, canvas, WebGL and X3DOM relation [27]

Video Tracking & 3D Visualization Web Application

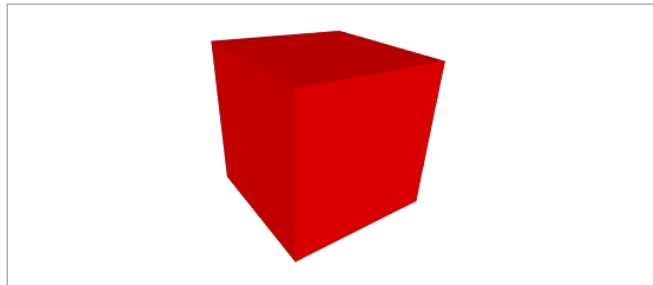
The name X3DOM [25] is a result of the combination of two brevities. The first, X3D ("Extensible 3D Graphics"), suggests a standard for 3D graphics. The second is DOM ("Document Object Model"), which describes the interactions and representations associated with the content of HTML documents [25]. X3DOM works as a description language for the 3D content in a Web page, as ready to use HTML tags. X3DOM elements are accessed through DOM operations, just like native HTML elements. Using X3DOM than other 3D libraries has several advantages [25] like:

- No plugins,
- Parts compatible with new HTML profile standard.
- Big and vital community.
- No need to learn new API, only knowledge of HTML and DOM elements.

So, if someone wants to use X3DOM to build an application, all what will need is a text editor and a browser.

```
<x3d width='500px' height='400px'>
  <scene>
    <shape>
      <appearance>
        <material diffuseColor='1 0 0'></material>
      </appearance>
      <box></box>
    </shape>
  </scene>
</x3d>
```

3.9 X3DOM code in HTML [27]



3.10 X3DOM rendered result [27]

There are several applications, with most of them be open-sourced. The most known examples of these applications are Blender [26] and the Sun Microsystems's Project Wonderland [27].

3.2.5 Pros and Cons

Babylon.js and X3DOM can be characterized more like platforms, having their own structure, with a lot of processing levels of a 3D model, until its ready to use in WebGL or GPU. As a result of this, they are easier-to-use for the engineer, but they present a limited performance when 3D models are big and complex. Three.js can be characterized more like an API or a library set, a thing that puts it closed to WebGL. It requires a more demanding usage for the engineer and uses a highly programmatic approach of 3D model integration. Although, it shows a really good performance for big and complex 3D models. PlayCanvas used the definition of game engine to be characterized, so that puts it closer to Babylon.js

and X3DOM, something which can be perceived from the similarity of their processes. PlayCanvas differs on the point that can handle a bigger and more complex 3D models, because of the usage of its own 3D editor. But still can't reach the level of Three.js performance. In this thesis we chose to use Three.js, because we were more interested in the performance of the application.

3.2.6 *Software Wrapper*

The Wrapper is a software design pattern (also known as Adapter pattern) that allows an existing program to be executed under another interface. It is commonly used to make existing programs cooperate with other without touching any source code. It's used to create flexible and reusable object-oriented applications. A wrapper function in a software library or a computer program has main purpose to call a subroutine, or do a system call with little or no addition of code [28]. As a wrapper, we can define an entity that encapsulates and hides the underlying complexity of another entity with a set of well-defined interfaces. Wrapper libraries [29] contain a small part of source code that converts an existing interface into a compatible to the technology used interface. This is done for several reasons:

- Improve an Interface
- Combine code parts that couldn't work together in other way
- Enable cross language and runtime interoperability

In a few words a Wrapper converts one interface to another so that it matches what the client is expecting. This is very useful if someone wants to combine different technologies in a project. For example, to take advantage of C++ speed and potentials, into a web project, a completely different environment for C++ to work. With the usage of Wrapper, C++ code is encapsulated in it and communicates with the web technologies part in a common language.

4 *Methodology & Implementation*

4.1 *Main Idea*

The main idea of this thesis is to produce a web application with the main function of tracking players positions and converting them into 3D world coordinates, so they can be visualized into a 3D world environment. This procedure is designed for sports events videos, in our case football videos. The main goal of this application is to provide users the ability of doing the tracking and visualization process by using their own browser. That's the major difference from the already existing solutions, the ability to execute this process wherever the user wants without any preparation action needed except connecting to the internet and having a video to process. The process of extracting positions can be accomplished in any video recorded by a conventional camera, without the need of specific equipment. The supported video format is mp4, this restriction is put in order to ensure compatibility with the HTML5 supported video formats.

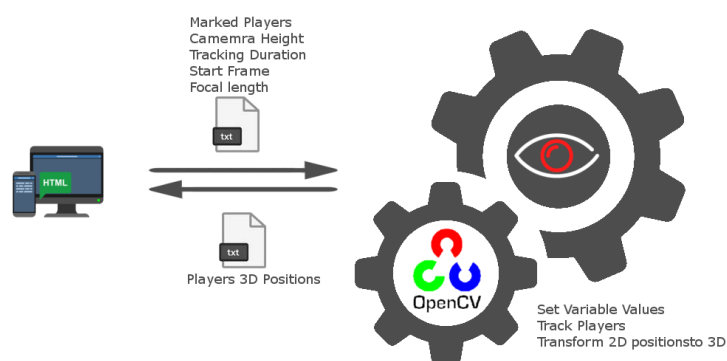
4.2 *Workflow & Difficulties*

The whole project started from a basis of a tracking algorithm implemented in C++. Fast and reliable algorithm but not as versatile as JavaScript algorithms. We came up with the question if we could implement this algorithm completely built JavaScript and follow a completely web technology approach in the project. After a small research on what libraries or frameworks were available on video and object tracking the results showed that there were pretty good libraries with OpenCV [12] and Tracking.js [13] look the most reliable.

Unfortunately, as most of the libraries found their approach was closer to face and color detection and recognition. Any implementations close to the main problem were in a very primitive stage, something that wasn't helpful. We also tried to build our own JavaScript tracking algorithm, but it didn't have the precision we needed. Under those restrictions, the final decision was to keep the C++ algorithm and run it as a service on web. This was achieved by using a software wrapper to make the algorithm friendlier to the web.

4.3 The C++ Tracking code

The Tracking code is based on the source code of the paper "Converting 2D motion into 3D world coordinates in the case of soccer players video" [6], which was transformed in the project's requirements, in order to have accurate transformations from 2D to 3D positions. The tracking process makes use of KCF algorithm to track the positions of players on the screen. Lets have a closer look to the source code works.

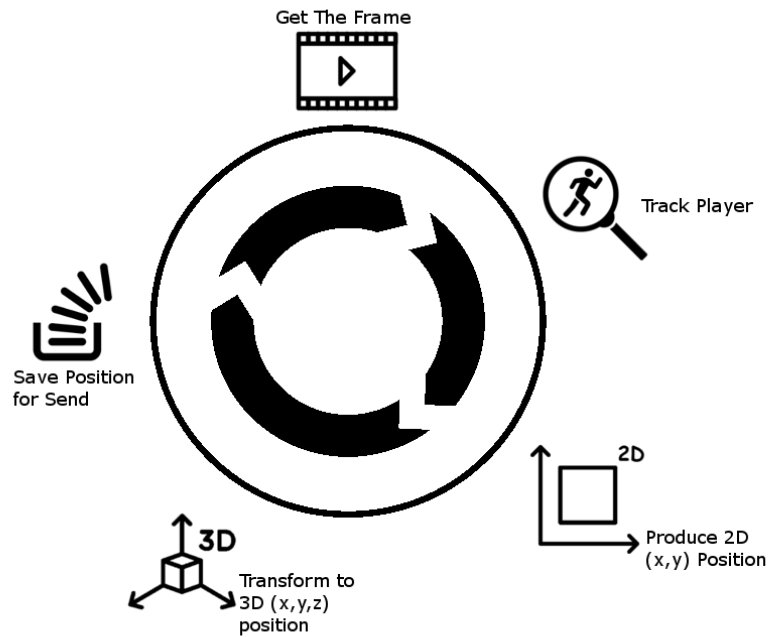


4.1 Algorithm Execution Process

The user, through the HTML interface, provides algorithm a file with the variables to be used for the tracking process. Algorithm set values to the variables and starts the tracking process. More specifically, the values that sent from user are:

- the marked players rectangles in a way that OpenCV understands them
- camera height
- Tracking process duration
- The frame from which the process will start
- Focal length of camera

After the algorithm variables get those values, the process starts and exports the 3D positions of the players.



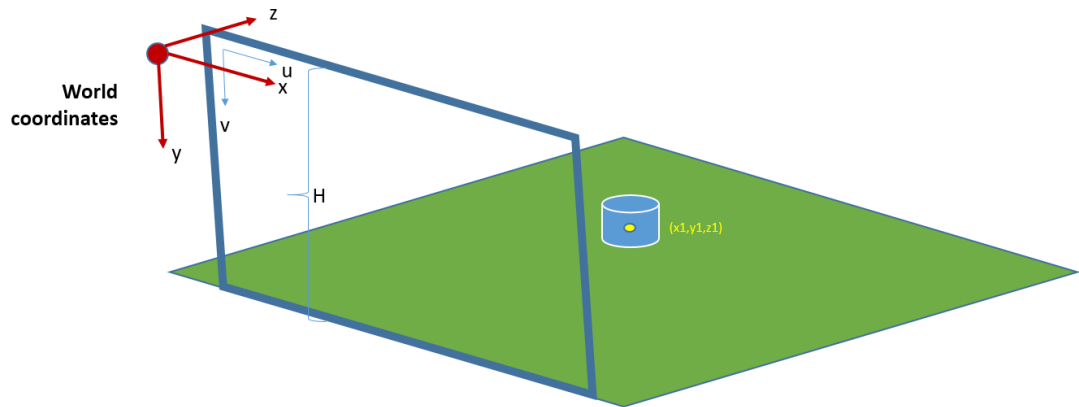
4.2 Tracking Process Workflow

The algorithm is fed with frames and for each one does a scan and locates the position of tracker's rectangle center on screen. Then this position is transformed into 3D by using 3D projection formulas.

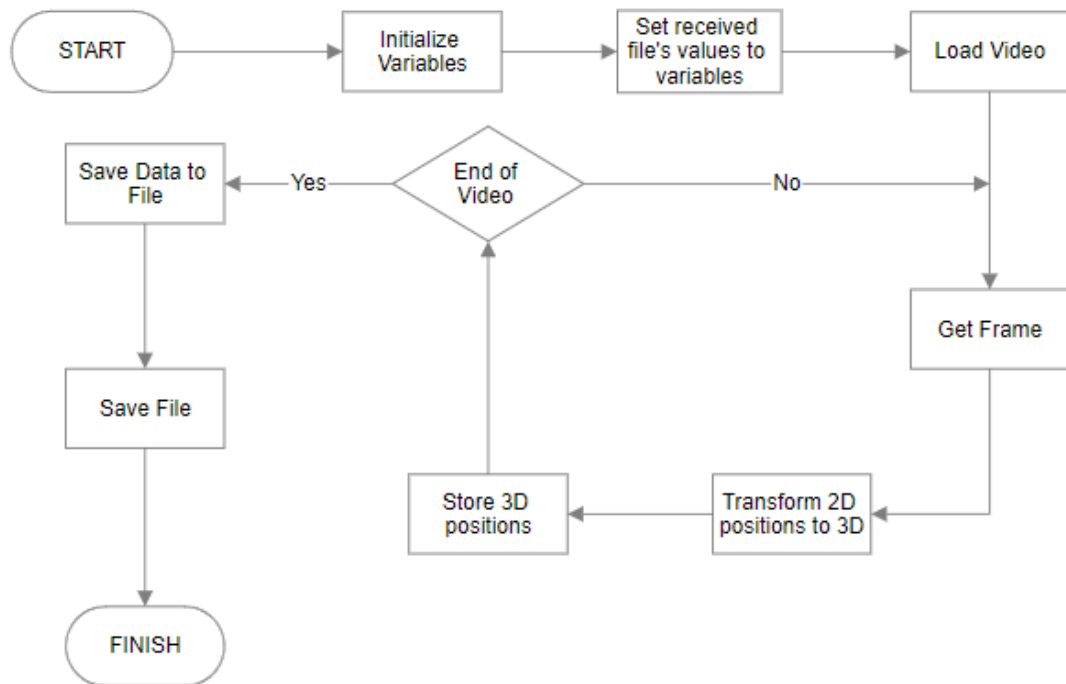
$$y = H \text{ (static)}$$
$$z = \left| \frac{y * f}{V} \right|$$
$$x = U * \frac{|z|}{f}$$

4.3 3D Transformation formulas

Video Tracking & 3D Visualization Web Application



4.4 Simplified Transformation from 2D to 3D coordinations

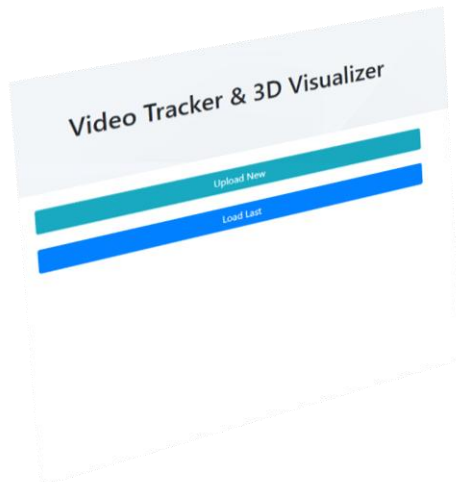


4.5 Server algorithm workflow diagram

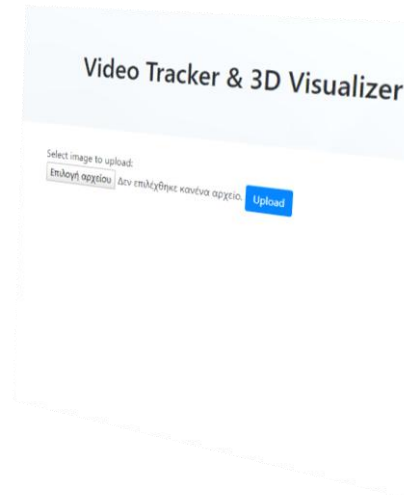
The tracking duration depends on the framerate that tracking process uses and not on video's. So, the 2 seconds tracking process that sent from user (who works at a framerate of 20-30 fps) would last longer for the algorithm depending on the number of players-rectangles has to track (average works around 5fps for 2-3 player rectangles). After tracking finishes the saved position data are encoded to JSON format and sent to user as file as data for the vizualization.

4.4 The App Interface

The user (client) side makes use of state-of-the-art web technologies. A PHP page with support of the latest elements as video, canvas etc., with Bootstrap 4 CSS for the optical beauty result and JavaScript (Vanilla.js) for the functionality. The whole website consists of 4 PHP pages, index page, the video upload page, the tracking page and the visualization page, each one with its own style and functions.



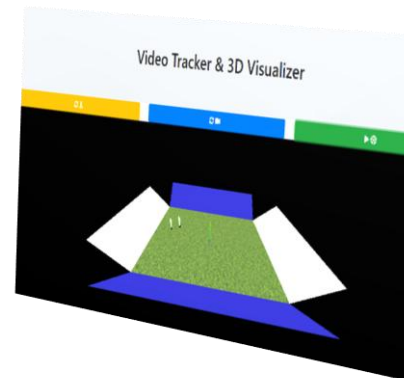
4.6 Video Upload Page 1



4.7 Video Upload Page 2

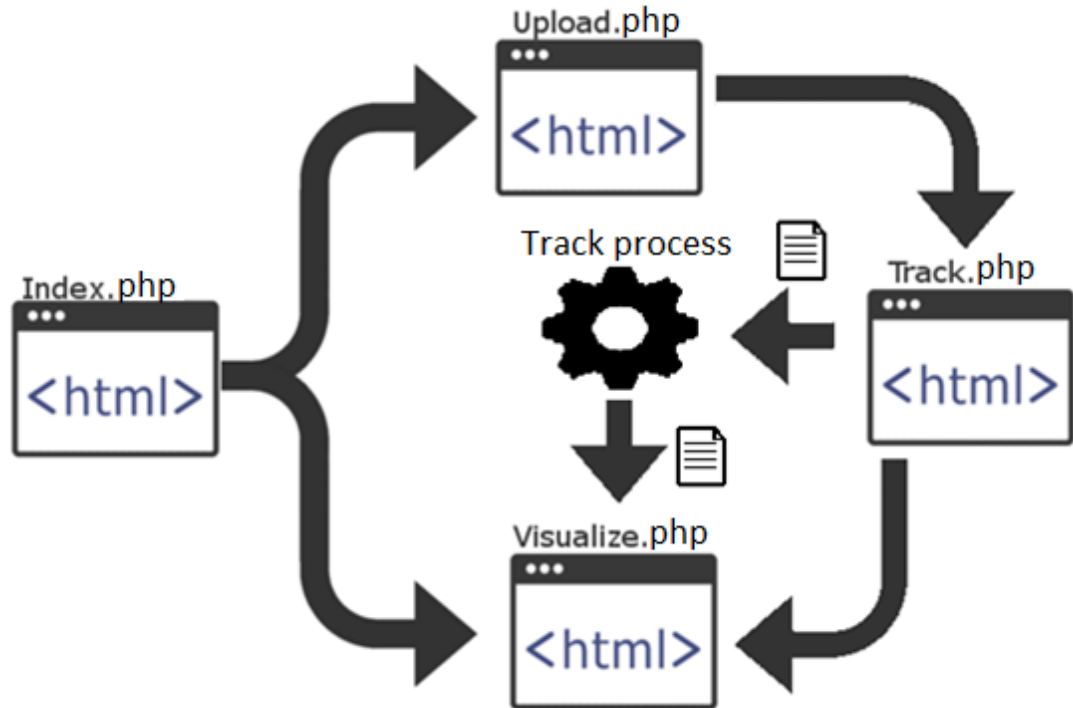


4.8 Tracking Page



4.9 Visualization Page

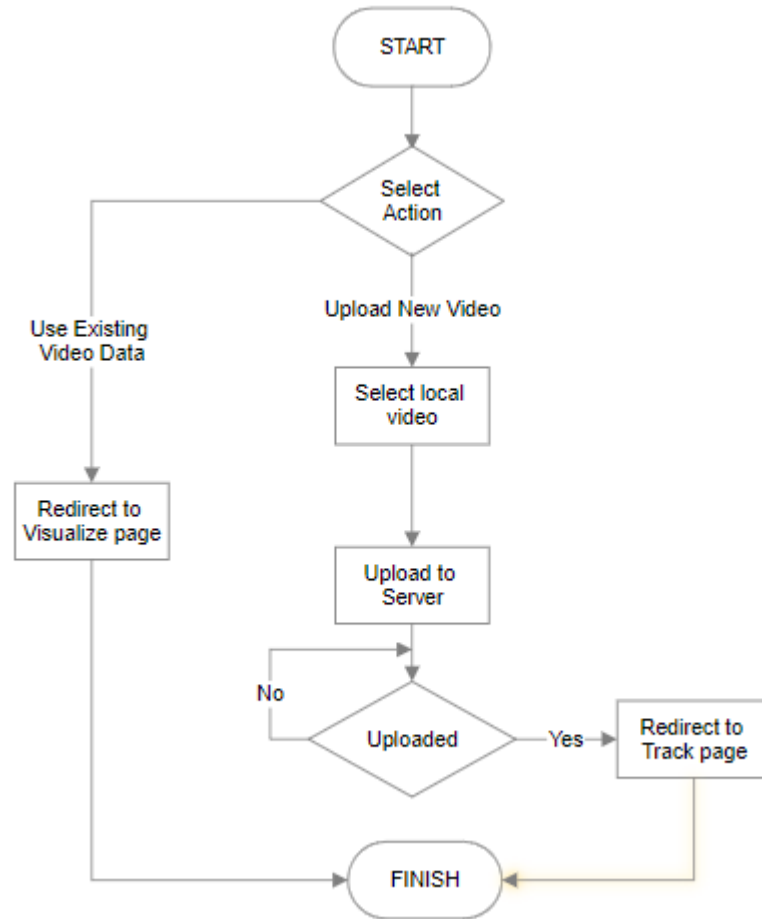
The index page lets the user choose between starting a new tracking process in a video that he uploads and visualize the last tracking process results. Depending on what he will choose he will be redirected either on video upload page to select and upload a new video for tracking from his local filesystem, or on visualization page to display the tracking results in a 3D representation. The tracking page shows the user the video he chose, which he can play or pause whenever he wants, until he reaches the spot that he wants to mark the players he wants and send the data to the server for the tracking.



4.10 Website Navigation Diagram

Let's see more extensible the two basic pages operations. Tracking page and Visualization page. In the tracking page as we mentioned before provides user an interface where he sees a video and marks on it the player he wants with rectangles. Because marking things on a video is impossible, as the video plays, we draw each frame on a canvas where we have the freedom of drawing things. So, the user marks the player he wants and saves them in a file in order to be transferred to the server. In the same file are saved some more data info, to help tracking algorithm be more accurate on his results for the specific video that user has in process. These data are the camera height, the camera's focal length and the time duration of the tracking process he wishes to execute. Things that vary depending on the video. When all these are saved in the file, this is sent to the server for the main tracking process.

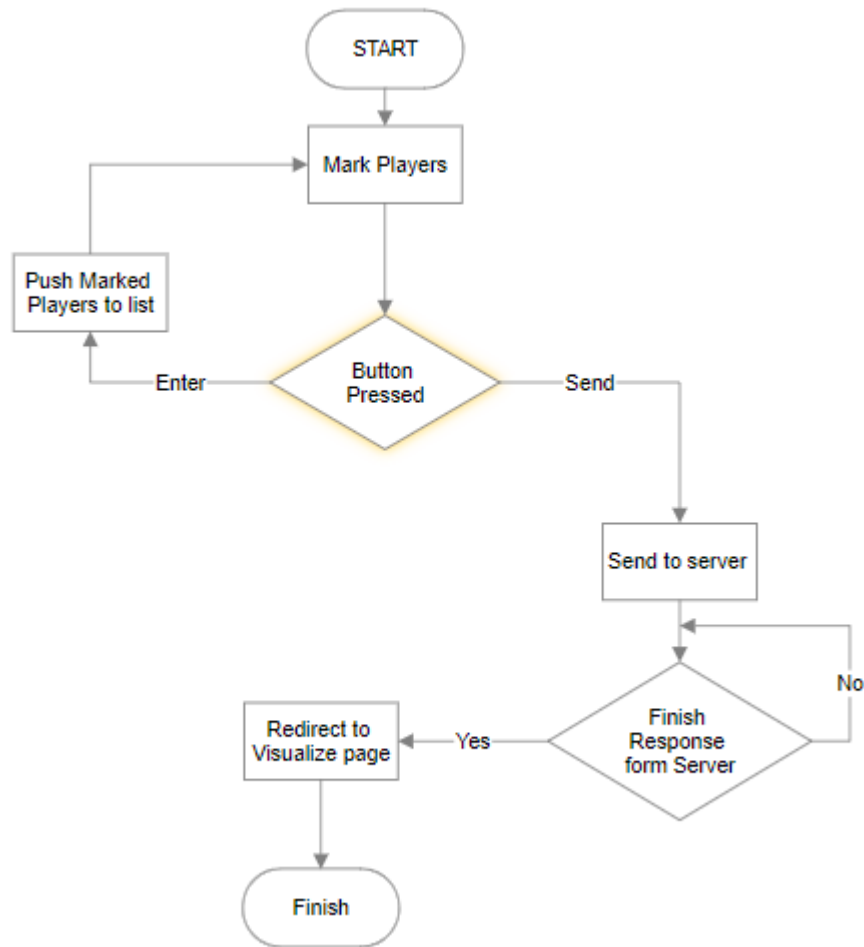
Video Tracking & 3D Visualization Web Application



4.11 Upload Video workflow diagram



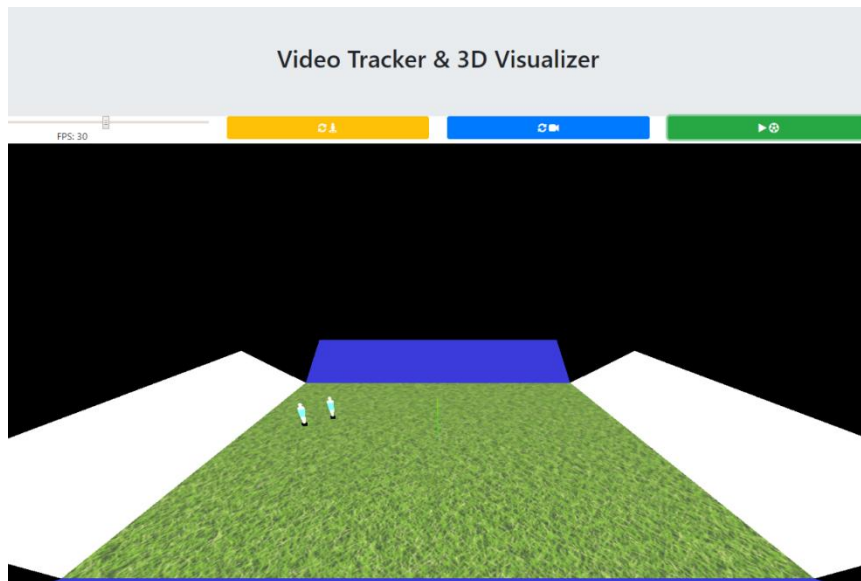
4.12 Tracking page Usage



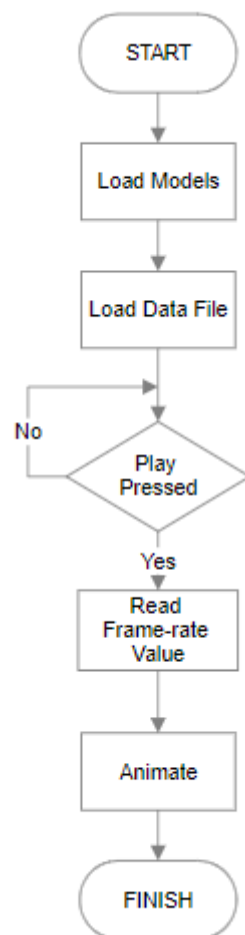
4.13 Track workflow diagram

The visualization page appears to the user after the tracking process finishes and server sends back the results response. The page loads the response file with the positions data and draws them into a 3D configured canvas, using Three.js library. It has a simple control panel from where user can choose the fps rate, start the animation, reset the camera at initial point and reset the positions and start over animation. The processing functionality is built completely in JavaScript with usage of Three.js library. The models and data are processed really fast and thing that zeros the loading time. Very simple and low-poly 3D models are used for much easier desired outcome. Concerning the animation, with the usage of requestAnimationFrame function of JavaScript and a framerate control code snippet the animation flow can be as smooth as user wishes to.

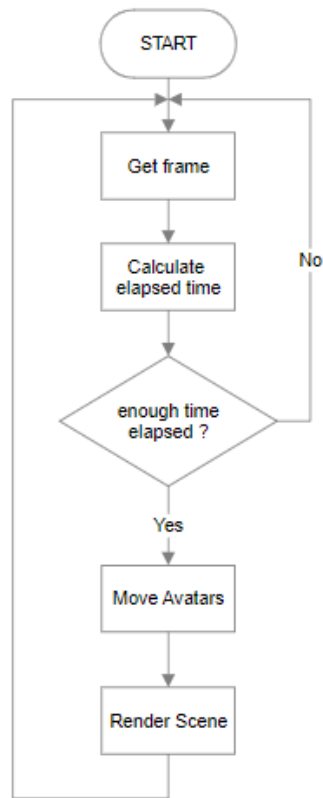
Video Tracking & 3D Visualization Web Application



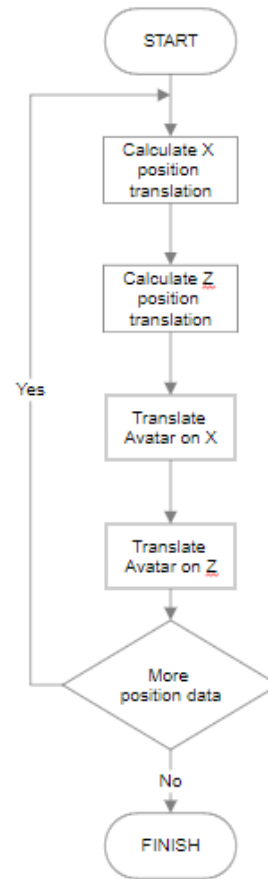
4.14 Visualization Page Usage



4.15 Visualization page workflow diagram



4.16 Animate function workflow diagram



4.17 MoveAvatar function workflow diagram

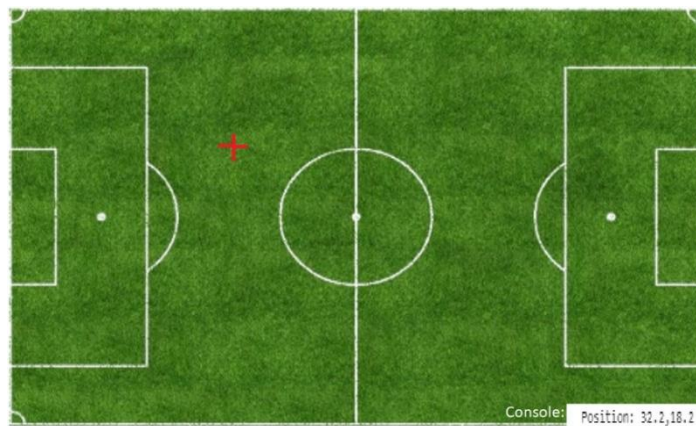
4.5 Sessioning

As the project’s implementation progresses the problem of multiuser accessibility arises. There should be users’ files isolation to avoid conflicts that may lead to data corruption of users in case that there is parallel execution of two or more tracking processes. A lot of techniques were examined with sessioning appearing as the most suitable for the project’s purpose. With sessioning each user has his files isolated, making use of the session identifier that each user’s browser creates by connecting to the application server. This identifier accompanies the files created and edited from the user (video & data files). The tracking algorithm benefitting from sessioning can now run and service more users by creating an instance of himself for each user demands his services. This eliminates the case of conflict between users and gives the application a higher level of credibility and integrity of the results.

5 Experimental Results

5.1 Experiments Execution

A series of experiments were executed to test the results that algorithm produces taking samples from a specific football match, with as much ideal conditions for the algorithm as could be. These conditions were a steady camera that covers as much area of the field is possible and for ease of comparisons between real position and 3D position, a pitch with already known dimensions. In experiments' case a pitch of 100m x 60m dimensions. Below are presented some key results of algorithm's experimental execution, compared with the actual approximate position in real field. As zero point for distances, the nearest left corner of the pitch in front of the camera is set. Positions are estimated based on a custom pitch position calculator.



5.1 Position Estimation Calculator

Position on pitch (m) Start: (36, 41) Finish: (32, 34)	3D Position(m) Start: (40, 41) Finish: (35, 36)
Deviation (m) Start: (4, 0) Finish: (3, 2)	

5.2 Experimental Example 1

	
Position on pitch (m) Start: (50, 30) Finish: (47, 24)	3D Position(m) Start: (47,49) Finish: (43, 48)
Deviation (m) Start: (3, 19) Finish: (4, 24)	


5.3 Experimental Example 2

The first two examples are examining the case of tracking only one player each time. The execution time is fast, and the results are more or less expected. Small deviation in X axis and big in Z axis (depth) because of the tracking rectangle with static height.

	
Position on pitch (m) P1 Start: (25, 18) Finish: (23, 9) P2 Start: (27, 43) Finish: (27, 35)	3D Position(m) P1 Start: (33, 40) Finish: (30, 40) P2 Start: (36, 54) Finish: (34, 53)
Deviation (m) P1 Start: (8, 22) Finish: (7, 31) P2 Start: (9, 11) Finish: (7, 15)	

5.4 Experimental Example 3

In the case of two players the only thing that had a sensible difference was the execution time. The movement of the players was for a long time along the X-axis, so a smaller deviation in Z-axis is observed.

	
<p>Position on pitch (m)</p> <p>P1 Start: (47, 22) Finish: (40, 7) P2 Start: (36, 43) Finish: (29, 36) P3 Start: (29, 24) Finish: (28, 9)</p>	<p>3D Position(m)</p> <p>P1 Start: (45, 51) Finish: (45, 52) P2 Start: (37, 51) Finish: (38, 52) P3 Start: (38, 41) Finish: (37, 41)</p>
<p>Deviation (m)</p> <p>P1 Start: (2, 29) Finish: (5, 45) P2 Start: (1, 8) Finish: (9, 16) P3 Start: (9, 17) Finish: (9, 32)</p>	

5.5 Experimental Example 4

The same with three players. The execution time increased a lot and the danger of tracker's overriding was sensible, although without losing contact with the initial target.

	
<p>Position on pitch (m)</p> <p>P1 Start: (42, 14) Finish: (38, 7) P2 Start: (33, 12) Finish: (35, 15) P3 Start: (32, 21) Finish: (29, 16) P4 Start: (30, 24) Finish: (24, 12)</p>	<p>3D Position(m)</p> <p>P1 Start: (54, 50) Finish: (45, 50) P2 Start: (40, 54) Finish: (42, 54) P3 Start: (38, 51) Finish: (39, 49) x (lost) P4 Start: (36, 51) Finish: (38, 49)</p>
<p>Deviation (m)</p> <p>P1 Start: (12, 36) Finish: (7, 43) P2 Start: (7, 42) Finish: (7, 39) P3 Start: (6, 30) Finish: (10, 33) P4 Start: (6, 27) Finish: (14, 37)</p>	

5.6 Experimental Example 5

Video Tracking & 3D Visualization Web Application

In the case of four players the execution time is just too long. The rectangle override happened for one player, when the tracker override with another player and followed other target. So, this calculation can't be deserved as trustful. Below are some more experimental examples results for 10 video samples.

Position on pitch (m)	3D Position(m)	Deviation (m)
P1 Start: (63, 25) Finish: (55, 12) P2 Start: (39, 39) Finish: (38, 27) P3 Start: (38, 22) Finish: (33, 8)	P1 Start: (53, 50) Finish: (52, 49) x P2 Start: (44, 44) Finish: (37, 43) P3 Start: (41, 55) Finish: (39, 54)	P1 Start: (10, 25) Finish: (3, 37) P2 Start: (5, 5) Finish: (35, 15) P3 Start: (3, 33) Finish: (6, 46)
P1 Start: (60, 38) Finish: (50, 32) P2 Start: (51, 42) Finish: (40,34) P3 Start: (35, 27) Finish: (35, 21)	P1 Start: (50, 47) Finish: (48, 44) x P2 Start: (44, 42) Finish: (41, 40) P3 Start: (41, 53) Finish: (38, 52)	P1 Start: (10, 9) Finish: (2, 12) P2 Start: (7, 0) Finish: (1, 6) P3 Start: (6, 26) Finish: (3, 31)
P1 Start: (41, 21) Finish: (36, 4) P2 Start: (55, 30) Finish: (46, 16)	P1 Start: (47, 48) Finish: (41, 47) P2 Start: (40, 53) Finish: (35, 53)	P1 Start: (6, 27) Finish: (5, 43) P2 Start: (15, 23) Finish: (11, 37)
P1 Start: (68, 16) Finish: (61, 5) P2 Start: (58, 15) Finish: (50, 3)	P1 Start: (59, 52) Finish: (61, 48) x P2 Start: (54, 55) Finish: (50, 53)	P1 Start: (9, 36) Finish: (0, 43) P2 Start: (4, 15) Finish: (0, 50)
P1 Start: (54, 45) Finish: (44, 29) P2 Start: (39, 39) Finish: (38, 27) P3 Start: (33, 28) Finish: (36,16)	P1 Start: (47, 41) Finish: (41, 38) P2 Start: (41, 48) Finish: (36, 46) P3 Start: (41, 54) Finish: (37, 54)	P1 Start: (7, 4) Finish: (3, 9) P2 Start: (2, 9) Finish: (2,19) P3 Start: (8, 26) Finish: (1, 38)
P1 Start: (44, 43) Finish: (50, 32) P2 Start: (33, 31) Finish: (40,34) P3 Start: (46, 19) Finish: (35, 21)	P1 Start: (50, 47) Finish: (48, 44) x P2 Start: (44, 42) Finish: (41, 40) P3 Start: (41, 53) Finish: (38, 52)	P1 Start: (10, 9) Finish: (2, 12) P2 Start: (7, 0) Finish: (1, 6) P3 Start: (6, 26) Finish: (3, 31)
P1 Start: (57, 27) Finish: (31, 30) P2 Start: (56, 28) Finish: (32, 27)	P1 Start: (50, 46) Finish: (50, 49) x P2 Start: (49,48) Finish: (49,48) x	P1 Start: (7, 19) Finish: (19, 19) P2 Start: (7, 20) Finish: (17, 21)
P1 Start: (57, 27) Finish: (31, 30) P2 Start: (40, 24) Finish: (25, 24)	P1 Start: (50, 46) Finish: (49, 42) P2 Start: (43, 51) Finish: (44, 47)	P1 Start: (7, 19) Finish: (17, 12) P2 Start: (3, 27) Finish: (19, 27)
P1 Start: (65, 9) Finish: (50, 15) P2 Start: (53, 15) Finish: (37, 15) P3 Start: (44, 6) Finish: (26, 10) P4 Start: (32, 9) Finish: (24, 5)	P1 Start: (57, 54) Finish: (55, 49) P2 Start: (49, 49) Finish: (49, 46) x P3 Start: (45, 54) Finish: (44, 51) x P4 Start: (40, 55) Finish: (39, 53) x	P1 Start: (8, 45) Finish: (5, 34) P2 Start: (4, 34) Finish: (12, 31) P3 Start: (1, 49) Finish: (18, 41) P4 Start: (8, 46) Finish: (15, 48)

5.7 Experimental Examples Table

The above experiments were executed on various video samples of same length of 10 seconds. The results that are marked with the red X (**x**), are marked as unreliable because for some reason the tracker lost the object during the process. Either because of faster movement that tracker can perceive, or because of losing focus due to passing behind other object that cuts tracking process of the initial object.

5.2 Results Evaluation

After the experiments execution completed some issues arise. Even the algorithm was fast and reliable, we faced the problem of big deviation on the depth axis due to the absence of a tracker responsive to the size change of the object. The fixed size tracker gave us zero or to small translation to the depth axis, because the movement on this axis has to do with the height of the object, according to the transforming formula. Another issue that appears is the case that the camera moves and changes pointing target. This has as a result an inconformity of the positions calculated where there was a movement of the zero point of video (hypothetic the center of the field in the video) and the positions at the 3D world field where the zero point is steady. That shows up a new problem that camera should be placed at the center of the field level and cover as much as its possible of the field's length. A problem that depends on the video's quality and the movement of the objects in it, is the tracker's override. If the video has low quality and the image is blur or the tracked players cross each other, there is a conflict on the trackers' data. This results in the trackers either stick together and follow the same object in case of override or lose the tracked object and stop in one place. Generally, it's an algorithm needs some optimization in those cases, but it's very reliable on the results as the video we have gets closer to the ideal conditions. These conditions are that the video is recorded with clear picture, a camera placed on the field's center level, pointing the center of the field, has wide angle of view and covers all the length of the field and doesn't move or rotate during the recording.

6 Conclusions

The purpose of this thesis was to create a web application of tracking players positions from soccer videos, exporting their positions and representing them in a 3D world environment. A procedure that will be accessible for all, either professional or amateur users that want to do this process from wherever they are, by accessing from their own browser. The implementation of this project was accomplished by using C++ source code with usage of OpenCV library and the KCF algorithm for the tracking process part. For the web interface, state of the art web technologies was used such as HTML5 and JavaScript. Its designed in a way to simplify the ease of access either for a team coach who wants to see how his team's players respond tactically to various situations, or simple users who want to see how their favorite players move and react. Even the players themselves can see how they react in some situations that maybe were mistaken or had wrong reaction. Furthermore, this web application can be used by the reporters that cover a football match or write their opinion about it, to provide a more valid and powerful article with enhanced replays and statistical analysis, helping them also strengthen their opinion and point of view.

6.1 Future Work

It is a project that still needs a lot of work and improvements to be done, in order to be a sufficient and trustful application for massive usage. Some of the improvements that need to be done are the reexamination of the transformation formulas to see if we can have more accurate results on positioning, the usage of more detailed graphics and animations and maybe the option of tracking and visualizing in real-time as much this is possible. But as we saw above in the experimental results the most urgent improvement is the depth positioning. This can be done by improving the tracker's box or tracker's rectangle as it's known in OpenCV. According to the Euclidian distance the position in depth of an object is relative to its height, as seen from the camera. This means that the closer the object is to the camera, the bigger it will be shown, so the further the smaller. As the tracking process is performed with static height rectangles, there is no proper precision on positioning results, because it's based only on the movement of the rectangle on this axis and not its size, which would provide much more accuracy. Therefore, that's the reason of the big deviation observed on this axis in experimental results. So, the usage of dynamic height rectangles based on object's size in each frame would be an improvement of major importance.

References

- [1] "Sentio Sports Analytics," [Online]. Available: <https://sentiosports.com/sports-solutions.html>.
- [2] "Sportcast," [Online]. Available: <https://www.sportcast.de/>.
- [3] "DFL Digital Sports," [Online]. Available: <https://www.dfl-digital-sports.de/>.
- [4] "Opta Sports," [Online]. Available: <https://www.optasports.com/>.
- [5] "lusob.com," [Online]. Available: <http://lusob.com/2012/02/tracking-a-football-match-with-html5-and-javascript/>.
- [6] C. Eirini, "Converting 2D motion into 3D world coordinates in the case of soccer players video," p. 70, 2017.
- [7] "kalman filter wiki," [Online]. Available: https://en.wikipedia.org/wiki/Kalman_filter.
- [8] "KLT wiki," [Online]. Available: https://en.wikipedia.org/wiki/Kanade%E2%80%93Lucas%E2%80%93Tomasi_feature_tracker.
- [9] B. Babenko, S. Belongie and M.-H. Yang, "Visual Tracking with Online Multiple Instance Learning," p. 8.
- [10] B. Martinez, M. F. Valstar, X. Binefa and M. Pantic, "Local Evidence Aggregation for Regression Based Facial Point Detection," p. 16.
- [11] J. F. Henriques, R. Caseiro, P. Martins and J. Batista, "High-Speed Tracking with Kernelized Correlation Filters," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, p. 14, 5 November 2014.
- [12] "OpenCV," [Online]. Available: <https://opencv.org/>. [Accessed 19 June 2018].
- [13] "Tracking.js," [Online]. Available: <https://trackingjs.com/>. [Accessed 19 June 2018].
- [14] "Intel Wikipedia," [Online]. Available: <https://en.wikipedia.org/wiki/Intel>. [Accessed 19 June 2018].
- [15] "Willow Garage Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Willow_Garage. [Accessed 19 June 2018].
- [16] "Emscripten," [Online]. Available: <http://kripken.github.io/emscripten-site/>. [Accessed 19 June 2018].
- [17] "asm.js," [Online]. Available: <https://en.wikipedia.org/wiki/Asm.js>.
- [18] "WebAssembly," [Online]. Available: <https://webassembly.org/>.
- [19] "OpenCV.js Tutorial," [Online]. Available: https://docs.opencv.org/3.4.1/d5/d10/tutorial_js_root.html.

- [20] "HTML 5 wiki," [Online]. Available: <https://en.wikipedia.org/wiki/HTML5>.
- [21] "HTML video wiki," [Online]. Available: https://en.wikipedia.org/wiki/HTML5_video.
- [22] B. Shero, "Shero Commerce," [Online]. Available: <https://sherocommerce.com/html5-video-and-what-are-the-advantages-of-using-it/>.
- [23] "Three.js wiki," [Online]. Available: <https://en.wikipedia.org/wiki/Three.js>.
- [24] "PlayCanvas," [Online]. Available: <https://playcanvas.com/>.
- [25] "x3dom," [Online]. Available: <https://www.x3dom.org/>.
- [26] "Blender," [Online]. Available: <https://www.blender.org/>.
- [27] "Open Wonderland," [Online]. Available: <http://openwonderland.org/>.
- [28] "wrapper function wiki," [Online]. Available: https://en.wikipedia.org/wiki/Wrapper_function.
- [29] "Wrapper library wiki," [Online]. Available: https://en.wikipedia.org/wiki/Wrapper_library#C++_wrapper.
- [30] Z. Kalal, K. Mikolajczyk and J. Matas, "Forward-Backward Error: Automatic Detection of Tracking Failures," p. 4.
- [31] "Three.js," [Online]. Available: <https://threejs.org/>.
- [32] "babylon.js," [Online]. Available: <https://www.babylonjs.com/#specifications>.