Immersive Web: Markerless Detection and Virtual Content Enrichment of Billboards using Deep Neural Network Based Image Detection

1st Jonathan Kossick
Open Distributed Systems
Technical University of Berlin
Berlin, Germany
j.kossick@posteo.de

2nd Abdul Aziz Sediqi
Open Distributed Systems
Technical University of Berlin
Berlin, Germany

3rd Felix Bublitz
Open Distributed Systems
Technical University of Berlin
Berlin, Germany
f.bublitz@campus.tu-berlin.de

***Abstract*—The aim of this project was to investigate the current state of immersive web technologies including its possibilities, limitations and potentials based on the implementation of an actual case of application. The basic idea of the use-case was to create a web app that allows the user to get additional virtual content to a present billboard using AR technology. By pointing the camera of the device at a billboard the application detects the visible advertisement and shows matching content virtually placed in relation to the actual billboard. The user is then also able to interact with the shown content.**

One of the key challenges was the implementation of the markerless detection since current immersive web technologies do not provide adequate functionality. The markerless detection was realized using OpenCV and YOLO object detection models.

Finally the approach was tested on different web browsers and platforms. Possible limitations of the used web technologies and the approach implemented were evaluated. One main focus of the evaluation lay in the latency of the object detection and limitations due to restricted hardware performance on some devices.

***Index Terms*—markerless-detection, web technologies, augmented reality, yolo, tiny-yolo, object detection**

Project files: https://gitlab.com/jokober/webxr-project

I. Introduction

In recent years hardware that enables Virtual Reality (VR) and Augmented Reality (AR) applications became broadly available and more affordable to consumers. Associated with this progress several software development toolkits and frameworks for different platforms have been published. Popular examples are the Apple ARKit and the Google ARCore SDK. Both offer extensive functionality for the development of native VR and AR applications limited to their respective mobile platforms (iOS/ Android). However there are several efforts to enable AR and VR for web applications as a platform independent approach.

Frameworks like AR.js, Three.js and A-Frame already offer some functionality that can be utilized for the development of immersive web applications. With the early specifications of the WebVR API in 2014 and the superseding first WebXR API specification in 2018 there are efforts by the Immersive Web Working Group to consolidate uniform AR and VR API specifications for the web. However in terms of performance and functionality those approaches can not easily be compared to the more mature native solutions enabled by toolkits like ARCore and ARKit.

The aim of this project was to develop a more complex immersive web application based on a real use-case. In the implementation process the current state of immersive web technology development including its possibilities, limitations and potentials were investigated.

The basic idea of the use-case was to create a web application that allows the user to point the camera of a (mobile) device at a billboard. The application then detects the advertisement and shows additional content in relation to the position of it. The focus of the content enrichment were the following two cases:

- Advertisers that want to give further information to their target audience
- Users that want to do subvertising.[1]

Five different billboard advertisements from three different advertisers were selected to test the approach. The application should run on most web browsers and on different operating systems like macOS, iOS, Windows, Linux and Android.

II. Technologies

This paper bases on different technologies and frameworks to enable 3D rendering, displaying AR content and apply object detection using pre trained deep learning models. All used frameworks will be described briefly below.

A. A-Frame

Based on the information of the projects homepage, A-Frame is a JavaScript framework that is developed by Mozilla [1]. It allows the representation of 3D-objects in the web browser. It rest upon the 3D framework Three.js and is popular in the area of virtual reality web applications. 3D environments are defined over basic html elements. The anchor for all 3D

[1]https://en.wikipedia.org/wiki/Subvertising

content is a so-called a-scene. It defines the three dimensional room. In this room, it is possible to place different elements like 3D boxes, cylinders, videos e.g.. Using plugins like htmlembed the scene can be extended with the ability to render html content.

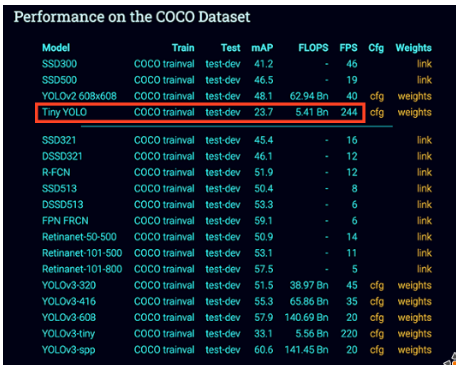
B. Darknet

Darknet is a popular open source neural network framework. It is written in C und CUDA and allows CPU and GPU computation. Darknet is used to train YOLO models [2].

C. YOLO and Tiny-YOLO

YOLO stands for *You Only Look Once* and is developed by J. Redmon and A. Farhadi [3]. It applies a single neural network to an image. The image is divided in multiple tiles. For each tile bounding boxes and probabilities are getting predicted.

In recent years there have been several revisions of YOLO including optimizations in model size, performance and accuracy. With Tiny-YOLO there is a variation of the YOLO architecture available that is less accurate but considerably smaller and faster. According to Redmon et al. Tiny-YOLO can achieve up to 244 frames per second on a single CUDA GPU [3]. Hence it is particularly applicable for applications where a high frame rate is required or for hardware with less performance.



Performance on the COCO Dataset

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20	cfg	weights

Fig. 1. Comparison of the performance of different object detection models on the COCO dataset [3]

D. OpenCV and opencv.js

OpenCV is a famous computer vision and machine learning library. It is open source and written natively in C++. There are several interfaces for C++, Python, Java and MATLAB [4]. With opencv.js there is a JavaScript binding that supports a subset of OpenCV functions. It is based on OpenCV 3.1.0. and leverages asm.js and WebAssembly [5], [6].

III. Related Work

As pointed out in the introduction there are two popular native toolkits for the mobile platforms iOS and Android. Both offer object detection and object tracking functionality. There are different projects und applications using those native technologies for applications similar to the described use-case [7]. However those are only targeting native platforms and are not running in web browsers.

In comparison there are only a few projects with the aim to run markerless immersive web applications with custom object detection. Some projects like AR.js leverage simple detection to enable marker-based recognition and positioning [8]. At the same time the number of frameworks and projects based on simple immersive web and WebXR technologies without object detection functionality is increasing. Some of the most popular ones are AR.js, Three.js and A-Frame.

Mozilla is one of the biggest driving forces for new immersive web technologies. They have not only created the A-Frame framework but are also pushing the WebXR API specification. In 2018 Mozilla Research Scientist Blair MacIntyre published a blog article called “Experimenting with Computer Vision in WebXR” [6] in which he describes his feasibility demonstration of computer vision on the web and also in the experimental WebXR polyfill environment [9].

In his article McIntyre mentions latency issues due to the lack of a video access feature in the MediaStream API. Furthermore he compares performance of the web and the experimental WebXR polyfill solution and latency issues due to asynchronously running opencv.js computer vision processing.

With JeelizAR there is one project which targets the same objective like this paper. JeelizAR is a JavaScript library that wants to bring real-time computer vision to developers. The library leverages a deep learning engine running on the GPU with WebGL. They claim to be able to do real-time face- and object detection as well as expression recognition.

The creators provide a few neural network models for object detection. However the range of available classes is very limited. Training of custom models is only available as a paid service [10]. As a demo they provide a web application which is showing an animated pot pouring coffee in a detected cup. While testing the detection of the cup was not reliable and the position did not update fast enough when moving the camera or the object leading to coffee being poured on the table (see figure 2). However the library offers interesting functionality for the implementation of immersive web applications running object detection on the actual device.

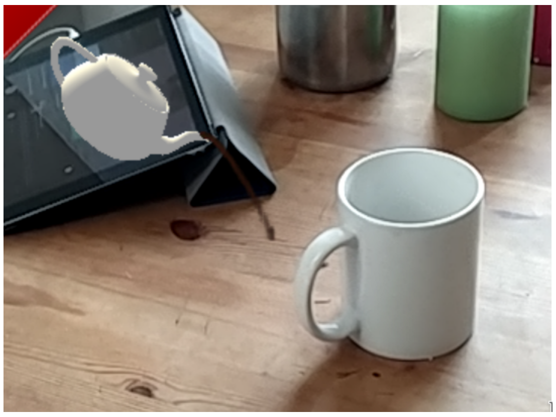


Fig. 2. JeelizAR demo web application running on Android

IV. Approach

The approach of the authors, documented in this paper, can be divided into two phases. In the first phase the image data of the camera is used to detect and categorize possible visible billboard advertisements. Subsequently the information of the detection phase will be used to position the AR data and provide it to the user. Both phases will be examined in the following sections.

A. *Advertisement Detection*

To recognize billboards and their position a supervised learning algorithm was chosen. Hence it is essential to train a model based on a labeled dataset in advance. Subsequently this model is used to determine the corresponding content for the visible billboard.

1) Training of the object detection model: For the training and validation dataset pictures of five relevant billboards from different locations were taken. Some of those images contained multiple target advertisements. With around 100 images the dataset was relatively small. Each image was labeled with the corresponding bounding boxes and names of the respective billboards. Since latency was crucial for the project the faster Tiny-YOLO model instead of the slower YOLO model was trained. Intensive drawbacks in accuracy were not expected because advertisements are relatively easy to detect because of their distinct composition. The Tiny-YOLO model was trained on the CPU using darknet.

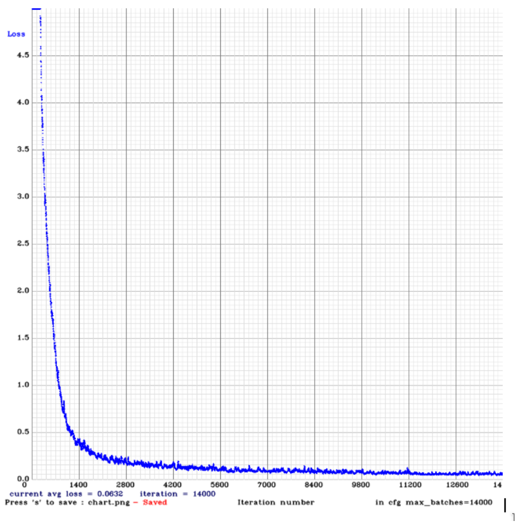


Fig. 3. Training loss-chart of the Tiny-YOLO model training on advertisement images

First tests of the trained model on single images were promising. All billboards were detected with confidences over 90%. The results of the detection on a test picture containing three different billboards are shown in figure 4.

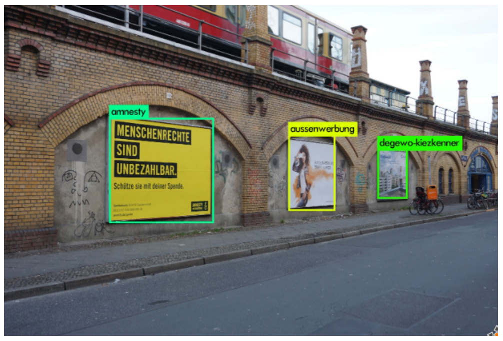


Fig. 4. Prediction result of the trained model on an image with three advertisements

2) Image Detection: The latency of the prediction is a major key-factor of the approach. Since computational power is significant for fast object prediction and due to the fact that the application should run on slower mobile devices, running the detection on the user's phone was not an option. Performance would be too slow on some devices and moreover the YOLO predictions do not run in the web browser. Therefore a server-side object detection architecture was used. This approach facilitates the use of faster hardware for the detection. In addition the application can run on devices with less performance.

If the user starts the web application a frame of the current visible camera stream will be sent to the server in regular intervals.

On the server side a Flask web server was implemented which receives images over HTTP PUT requests over TCP. The server runs the detection on each image and filters out all detections which are below the confidence threshold of 80%. Afterwards the OpenCV non-maximum suppression algorithm is used to consolidate all bounding boxes found for the same detected object to a single bounding box for each advertisement.

The detection information of those bounding boxes are then put into a JSON formatted response (see figure 5). The JSON contains the most important predictions including the labels, confidences and the bounding box coordinates. The response is sent back to the client where it will be used to load corresponding content and position it according to the bounding boxes detected.

B. *Display Content*

At that point it has been shown how advertisements are identified by the server. In this section it will be described how the user gets further information to an advertisement. This process of providing the consumer with added details to an advertisement can be structured in three phases.

The content provider has to define which information should be available as additional virtual content in advance. When the user points his device at a billboard the content must be loaded from the server. Finally the information must be provided to the user in an appropriate way. To achieve this

```
[{
  "label": "amnesty",
  "confidence": 0.9999868869781494,
  "position": {
    "x": 0.086,
    "y": 0.391},
  "size": {
    "width": 0.2285,
    "height": 0.27125}
},
{
  "label": "aussenwerbung",
  "confidence": 0.999350368976593,
  "position": {
    "x": 0.5676666666666667,
    "y": 0.46},
  "size": {
    "width": 0.2705,
    "height": 0.29375}
}]
```

Fig. 5. Server response containg prediction results

matter augmented reality seems to be the most user-friendly way. Meaning that the virtual interface creates the illusion that the additional content sticks right to the real content of the actual billboard. A two-layer architecture was chosen to reach that requirement, which consist out of the camera image as the lowest layer. On top of that a 2D transformation of a 3D room is placed using A-Frame, which allows displaying a variety of different content types like 3D-objects, videos e.g..

1) Two-Layer Architecture: As already mentioned a two-layer architecture is the basis of the AR experience. Therefore it is essential how the two layers are composite and positioned. Given the A-Frame as a 3D room and the camera image as a 2D plane. The idea is to define the virtual 3D room in a way that a plane at the position vector (0, 0, 0) with the width and height of 1 overlaps exactly the 2D device camera layer. Therefore the virtual 3D camera is placed on the position (0 , 0, -1) looking to the origin (0, 0, 0). The aspect ratio of the virtual 3D camera is defined as the aspect ratio of the device camera aspect ratio. With this definition every object can be placed in front of the device camera image using relative positioning. The described composition is illustrated in fig. 6.

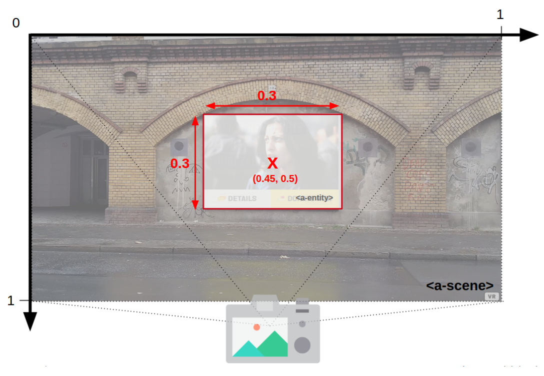


Fig. 6. Two-Layer Architecture

2) Content: The description and delivery of additional content is a crucial part for the quality of the solution. At first people are confronted with advertisements in an outside environment. Therefore most people using their phones in mobile networks like 3G or 4G and often have limited monthly bandwidth conditions. Combined with a real-world scenario where many different advertisements with partly big file sizes are available, there must be a focus on just loading the needed content and describing it in a dynamic but low bit-cost way. Supplementary it is important that the system is not restricted to specific content types and content providers are able to define the content in an easy way.

3) Data structure: To avoid the huge overhead to load the complete content of all available billboards at once, a list of available advertisements is divided from the actual content. This list (formatted as a json file) contains an identifier for all available billboards and a reference to the additional virtual content. If an advertisement has been detected the linked content json file will be loaded. It describes the additional data that should be displayed (see figure 7).

This content descriptor comprises a *name* attribute that specify the content provider campaign. A further attribute *views* is given as a list of virtual advertisement layers. An advertisement can consists of different views but just one view will be visible coincidently. At the moment of loading the advertisement content, the first available view will be displayed on the billboard. All other views are hidden and can be shown by a so-called change view action.

```
{
 "name" : "campaign name",
 "views" :[
   {
     "title" : "main view",
     "widgets":[]
   },
   {
     "title" : "details view",
     "widgets":[]
   }
 ]
}
```

Fig. 7. Virtual advertisement content

Each view is defined by a list of widgets, that describes the actual visible content. A widget can be a video, a button, html code, a 3D model, a menu or text. But it is also possible to create new widgets that represent other functionalities like a voting tool. Each widget is defined by a *type*, *position*, the *size* and other additional attributes like *video source* and *autoplay* of a video widget. The position and size is described relatively to the actual visible billboard. That means that a widget of type video with given x and y position of 0 and a width a height of 1 results in a billboard filling video. An example of a widget description is shown in figure 8.

4) User Interactions: Displaying additional digital content to the offline advertisements brings added value to campaigns. However the content without the ability to interact with it has a static nature. Providing the user the possibility to control the displayed content enhances the experience. To achieve this, a multi-page content architecture was designed. This

```
{
 "type" : "button",
 "position" : {
     "x" : 0,
     "y" : 0.8
 },
 "size" : {
     "width" :0.5,
     "height" :0.2},
 "background" : "#e0e0e0ee",
 "title" : "details",
 "icon" : "data/images/chat.svg",
 "action" : {
     "type" : "change_view",
     "value" : "details"
 }
}
```

Fig. 8. Widget example

infrastructure was already introduced in the previous section. With it the content provider gets a way to specify different content views and define transitions between it using *change view* actions. These actions are mainly triggered by touch gestures on widgets of the view. The touch gestures are recorded by an A-Frame raycaster. The following predefined actions can be executed by touching a button widget:

- open web link
- change view

5) Loading data: As already described in the section “data structure”, the real content of the billboards is divided from the list of available billboards to avoid loading unnecessary data. That results in less bandwidth use and a better performance. This concept of lazy loading data is an essential key-factor to enhanced performance. However not only the billboard content is affected by this improvement. Enabled by the abstraction of content types it was possible to load the required content type descriptions (widgets) on-demand. To also not reload images and videos of an already detected advertisement on a new recognition the resources are cached and stored using an unique identifier.

V. Evaluation and Discussion

The implemented approach allows the virtual enrichment of billboards using web technologies. Web technologies allows the theoretically possibility to reach every device that can interpret javascript. However there are different supported versions of javascript depending on the used web browser. Below it should be evaluated which browser supports the used frameworks. Keeping in mind that the major scenario is the using of the approach in mobile conditions, the bandwidth of the application and the latency should be particularly considered.

A. Supported platforms

The implementation was tested on different devices and web browsers. While the client-side object tracking brought trouble on different browsers except chrome the main features of A-Frame were supported on many tested platforms and browsers. Since the detection was outsourced to a server this approach is fairly supported on all modern devices and browsers. However since A-Frame based on Three.js there are minimum requirements. The browser has to support WebGL [11]. Additionally the fetching of the camera image using the MediaStream API has minimal browser version requirements. The minimal browser versions for the used features are illustrated below in table I.

TABLE I
Minimal requirements

Feature	Chrome	Firefox	Opera	Safari	IE	Edge
WebGL	56+	51+	43+	-	-	79+
MediaStream	14+	15+	66+	13+	-	18+
all	56+	51+	66+	-	-	79+

Source: https://caniuse.com/

B. Latency

In the implementation phase latency was identified as a crucial requirement for the application. Especially at the point of perceived usability the performance could be pivotal. If the camera of the device is not moving that fast the impression of a fluent interface can be kept although a reduced content position update frequency is present.

Overall latency in our approach is impacted by several factors including MediaStream API limitations, network limitation and performance of the object detection itself. Below the mentioned factors are particularly exemplified.

1) MediaStream API Limitations: To retrieve a single video frame from the MediaStream it is necessary to attach the media device to a DOM video element and afterwards it is rendered from the video element into a canvas element. The image is then extracted from the canvas. This procedure is computationally expensive and results in increased latency. This limitation of the MediaStream API is a limitation which affects a lot of applications connected to computer vision.

However there is a MediaStream Image Capture draft proposal which purpose is to more efficient, allowing developers to directly retrieve an ImageBitmap from a video stream using the grabFrame() method. It is currently only implemented in Chrome and hence was not used in this project.

2) Network Limitations: With the selected server-side-detection architecture better hardware performance can be used as running it on the client side. However at the same time the application becomes more susceptible to limited network bandwidth and latency. Table 1 shows how different network conditions can affect the overall performance.

The usage of networks slower than 4G will reduce the update frequency drastically. The tests have shown that on 3G networks the time between the request being sent by the client and the reception of the answer will be around 20 seconds.

However in future improvements of the application a new approach using WebRTC and UDP instead of an HTTP over TCP web server could provide promising improvements regarding network latency. Moreover with the upcoming spread

TABLE II
LATENCY TESTS

Bandwidth Throttling	**Bandwidth Test Results**[a]		
	Sending	***Waiting***	***Receiving***
2G	240 000 ms	120 000 ms	0 ms
3G	15 000 ms	5 000 ms	0 ms
4G / LTE	600 ms	70 ms	0 ms
DSL	0ms	36 ms	0 ms

[a] *Tested with Server running the Tiny-YOLO detection in OpenCV on AMD Ryzen™ 7 3700X CPU.*

of 5G in many countries the mobile network conditions will be improved noticeably.

3) Detection Speed: As shown in figure 1 detection performance highly depends on the selected object detection model. As it has been mentioned above the Tiny-YOLO model has been selected which is notably faster than other models. A drawback in accuracy was not perceivable.

Nevertheless there are possibilities to further improve detection speed drastically by using hardware which utilizes GPU computation for image detection. The impact of hardware performance was already noticeable when using different CPUs. On a 2.5 GHz Quad-Core Intel Core i7 notebook CPU running the detection in a low frequency of one FPS took around 0.03 seconds per image which is equivalent to 30 FPS. However running detection more frequently with 10 FPS resulted in a drastically reduced detection speed of 0.5 seconds which is equivalent to only 2 FPS. In comparison the creators of YOLO, Redmon et al., managed to achieve 244 FPS on a Pascal Titan X GPU [3]. Unfortunately the approach could not be tested to run on hardware with a CUDA GPU. Doing further tests on such hardware would be promising.

C. Object Tracking

Another promising improvement of the current approach could be to use a combination of server-side object detection and client-side object tracking. The advertisements would be detected by the server and the resulting border box information could be used to initialize a tracking algorithm running on the users device. This procedure would allow to increase the positioning frequency and reduce the bandwidth requirements significantly. However the opencv.js implementation of OpenCV is still missing some important tracking algorithms known from OpenCV. With the further development of opencv.js and better usage of WebAssembly the application could be improved noticeably.

VI. CONCLUSION

This paper stated how web technologies, object detection and augmented reality can be used to implement a markerless detection and enrichment of advertisements with additional information. The benefit of this approach is a platform independent, installation-free and fast recognition. It has been shown how a server-side detection approach enables deep neural network object detection for web applications even on devices with less performance. In combination with the A-Frame framework the advertisement can be virtually extended with data like video, 3D-objects e.g.. Therefore a bandwidth saving, highly customizable and abstracted way of defining the advertisement content was developed. Additionally touching actions were implemented to allow the navigation of the virtual content.

Testing the approach made different critical issues visible. For providing the user a fluent experience the accurate and fast detection and positioning of the content is essential. While the server-side detection architecture has many benefits there are some limitations that arise due to increased latency issues.

As a possible solution a hybrid approach might be promising. This would include less frequent object detections on the server side combined with more frequent client side object tracking for content positioning. Latency problems of the server-side detection architecture could be improved by this approach.

Apart from this it could be shown that web technologies qualify for the described use-case of markerless advertisement content enrichment. However in comparison to native frameworks, immersive web technologies are still very limited in regards of functionality and performance. Further development and enhancement of web technologies are necessary to enable the development of real use-case applications.

REFERENCES

[1] A-Frame: Introduction. Accessed: 2020-03-14. [Online]. Available: https://aframe.io/docs/1.0.0/introduction/

[2] J. Redmon. Darknet: Open Source Neural Networks in C. Accessed: 2020-03-14. [Online]. Available: https://pjreddie.com/darknet/

[3] J. Redmon and A. Farhadi. Yolo: Real-time object detection. Accessed: 2020-03-14. [Online]. Available: https://pjreddie.com/darknet/yolo/

[4] OpenCV: About. Accessed: 2020-03-14. [Online]. Available: https://opencv.org/about/

[5] S. Taheri *et al.* Introduction to OpenCV.js and Tutorials. Accessed: 2020-03-14. [Online]. Available: https://docs.opencv.org/3.4/df/d0a/tutorial_js_intro.html

[6] B. Macintyre. Experimenting with Computer Vision in WebXR. Accessed: 2020-03-14. [Online]. Available: https://blog.mozvr.com/experimenting-with-computer-vision-in-webxr/

[7] A. Morozova. Capabilities and Limitations of Apple ARKit and Google ARCore Anastasia Morozova. Accessed: 2020-03-14. [Online]. Available: https://jasoren.com/capabilities-and-limitations-of-apple-arkit-and-google-arcore/

[8] J. Etienne. AR.js - Augmented Reality for the Web. Accessed: 2020-03-14. [Online]. Available: https://github.com/jeromeetienne/AR.js

[9] WebXR polyfill with examples. Accessed: 2020-03-14. [Online]. Available: https://github.com/mozilla/webxr-polyfill

[10] JeelizAR - JavaScript/WebGL lightweight object detection and tracking library for WebAR. Accessed: 2020-03-14. [Online]. Available: https://github.com/jeeliz/jeelizAR#neural-network-models

[11] Three.js: Browser support. Accessed: 2020-03-14. [Online]. Available: https://threejs.org/docs/#manual/en/introduction/Browser-support