APPLYING MARKER-LESS AUGMENTED REALITY USING COMPUTER VISION

Shivank Yadav¹, Ayush Agarwal², M. Vaidhehi³
¹,² Dept of CSE, SRMIST.
³ Assistant Professor, SRMIST

ABSTRACT

The COVID-19 pandemic has brought concern in people’s mind to shop outdoors resulting in massive innovation in technology related to enhancing online retail experience. One such idea is using AR to enhance e-commerce visually. Many AR innovations still require a lot of hardware and operating expertise which is not available to customer at his home. Our aim is to provide a direction for someone who is imagining on creating mobile AR applications for the above use case. This paper presents a survey on recent technology of Computer Vision techniques and AR basics for implementing marker-less tracking for Augmented Reality applications.

Keywords- Computer Vision, Augmented Reality, feature extraction and landmark detection, static and dynamic gesture.

I. INTRODUCTION

User Experience is a very important factor of any software product and thus companies spend a lot of money in the research and development stage to make sure that customers get a smooth and hassle-free experience. In today’s age it is observed that in order to further improve user experience the use of Machine Learning is to be employed, this is simply because of the fact that the more amount to operations the computer is able to perform without user guidance the less work is to be done by the customer. We had this in mind and wanted to improve the user experience for applications which require hand detection, in all the existing models marker based tracking is implemented as it is easier and less costly to implement, but this causes a lot of trouble to the user as they have to use a marker and keep it properly as if it has wear and tear it won’t be recognized. Hence we plan on surveying all the possible ways to implement Computer Vision based marker less object tracking and integrate it with Augmented Reality Software Development Kits to create an interface ready for customers to use.

The object of choice for tracking here is a human hand and we have framed this survey based on a use case which is implementing augmented watch trying system. The main idea is to track hand landmarks using Computer Vision and use them as placeholders for AR watch object. The similar concepts can be used for variety of interactive AR applications.

II. LITERATURE REVIEW

“Researchers from Google [1] present an on-device a real-time solution predicting a human palm skeleton from a single RGB camera for AR products. The below 2 modals comprise their pipeline: 1) a palm detector responsible for bounding box generation across palm and provides it to, 2) a hand landmark model, that maps a palm sketch on the full hand. This is implemented through MediaPipe [2], ML solutions building platform. The model and pipeline proposed results in real-time inference speed on portable GPUs with good guess quality.

The figure 4 depicts the MediaPipe graph for the hand-tracking architecture. The graph can be divided into 2 subparts, one for hand detection and the other for points of reference mapping. A key optimization authors provide is that the hand detection only runs as needed, saving a lot of processing power. They accomplish this by extracting the palm location in the present video frame from the generated hand landmarks in the preceding frame, thus preventing the palm detection each frame. For toughness, the tracker even provides confidence score
for captured hand, whenever the confidence drops below the threshold then only the palm detector is reapplied to the upcoming frame.

![Image](image.png)

Figure 1: MediaPipe Graph of model implementation.

“Koller, Hermann and Bowden’s” [3] work shows a novel perspective to learning a classifier which is on the basis of frames on poorly differently marked sequence data by combining a Convolution Neural Net in an Expectation Maximisation algorithm. This permits the C.N.N. to be trained on a big set of trial images even if limited sequence level knowledge is available for the source videos This method is explained in situation related to hand shape detection, it could be of a wider use to any video based recognition task which doesn’t have frame-level labelling. The discriminative ability of C.N.N is used by E.M. algorithm to iteratively refine the frame level mappings and further training of the C.N.N. The C.N.N. can be easily trained on a million hand images by merging the classifier within E.M. architecture. They show that the output classifier model generalises over sets and entities. Data comprising 60 distinct classes of 3000 manually labelled hand pose images is used for evaluation.

Instead of using 2 stage architecture pipeline consisting of localizing hand and mapping landmarks, Pose Anchor [4] presents a single effective network architecture for hand landmarks detection. An end-to-end C.N.N. is trained on a newly proposed pose anchor network, which is based on RPN in Faster Region-based C.N.N [5]. Instead of manually designing hand pose anchors, they are generated using K-means clustering based on OKS [6], in a data driven way. We can get multiple representative pose anchors with various angles, hand signs and scales using this method The main reason for using pose anchors is to mitigate occlusion to some extent by using the prior knowledge of hand pose/structure Experiments conducted on (LSM-HPD) and NZSL were used to show the robustness and feasibility of the architecture.

Fig.2 shows the architecture of our model consisting a C.N.N with 2 branches. The input for training the net consists of hand images and pose anchors. The spatial location of a certain feature map is mapped to a source picture, and then all the anchors are placed to each position. After that, 0/1 labels are provided for binary
classification by assigning the pose anchors positive and negative according to similarity. The regression branched is supervised via the deviation computed between pose anchors and the ground truth hand landmarks. At the test stage, we generate the hand landmarks directly in the picture with the data of the 2 branches and the referenced pose anchor. Especially, the regression branch values matching to the highest classification points is taken when facing the restricted scenes with only one hand. Otherwise (N.M.S.) will be adopted.

For single-depth images “Robust 3D Hand Pose Estimation” is used. In these grayscale images depth is depicted in the form of pixel intensity change. Nearby objects have higher pixel values and are lighter than distant objects. These types of images are inputted by Microsoft Kinnect.

Fig. 3 A Single Depth Image with darker mannequin and lighter coloured vase.

This study showed that satisfactory performance hasn’t yet been achieved by most of the methods, due to the complexity of the problem. Most of the existing information driven methods cannot completely utilize the depth information because they firstly regress 3D hand pose from depth image estimation is proposed. The point cloud which is In this paper, they a novel multi-view CNN-based approach (Figure 4) for 3D pose estimation is proposed. The point cloud which is generated from the query depth image is projected onto multiple views of two projection settings for better exploitation of 3D information. Multi-view CNNs learn to associate projected pictures with heat maps which reflect the probability distributions in each view. After these maps have been fused to estimate optimal three dimensional pose with learned pose priors, a view selection method is used to suppress all the unreliable information. The results showed that this proposed method is better than the best methods on two challenging datasets. Furthermore, a cross-dataset experiment was carried out which validated that their approach had good generalization ability.

Fig. 2. Network Architecture. Pose anchors are placed on each green point in the input image.
Selection of Game Engine:
There are many game engines available to make AR applications. To chose one among them many factors must be kept in mind, they are:

Scripting: The code written to provide the instructions.

Rendering: The generation of the 3D scene, the factors to be measured are speed and accuracy.

Animation: the change in render per frame to simulate movement.

Artificial Intelligence: Ability of the computer to make decisions.

Physics: Real world based calculations on simulated physical interactions.

Audio: Audio feedback provided that can be spatially distinguished.

Networking: Users have the facility to play with others online.

A logical approach would be to go with a game engine which is widely used in the market, so we went through the survey of Brent Cowan and Bill Kapralos in which they have used 2 Surveys\(^7\) to determine a suitable game engine.

Survey 1: Scan through a database of approximately 200 academic publications for “serious game”, “educational game”, and “simulator”.

Survey 2: Narrow down these selected engines from Survey 1 only based on “serious gaming”.

There Final Results are shown in Table 1

<table>
<thead>
<tr>
<th>Game Engines</th>
<th>Unity</th>
<th>Unreal</th>
<th>H3D2</th>
<th>Mobile Engine</th>
<th>增资</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level editor</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Scripting</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>C++</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Networking</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>3D Graphics</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Shader effects</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Dynamic shadows</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Physics</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Artificial intelligence</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Free non-commercial</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Free for commercial</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Mobile Devices</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Web player</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
</tbody>
</table>

Table 1: Game engines and frameworks and their features.
Thus we can see that Unity and Unreal are the most beneficial game engines, we chose Unity as our preferred game engine as the only advantage of Unreal over Unity is C++ support, but the programming language to be used is not an restriction in our situation.

**Unity Engine Properties:**

Lighting can be done before hand or at run time, custom shaders can be created using shader graphs and shader programming thus replicating realistic lighting conditions as well as textures which can be changed based on input to change the simulation environment. Unity uses C# as the programming language behind the implementation of any application built using the framework and thus complicated logical simulations and inputs can be generated.

**The Machine Learning-Agents Toolkit provided by the game engine:**

This is an free to use SDK available to integrate ML models with Unity. The 3 main features in the SDK are Sensors, Agents, and Academy.[8]

The agents collect, observe and execute actions. The agent is the component being trained by constantly optimizing it’s policy known as Brain.

The Academy manages the simulation, it is a singleton and thus used to keep track of all the steps involved in the simulation.

Singleton is defined as a software design pattern that restricts the instantiation of a class to one "single" instance, any other instances which are created are automatically destroyed.

A Python package is provided which contains a class called UnityEnvironment containing the above mentioned components. Communication happens through a gRPC communication protocol.

![Diagram of Unity Environment](image)

**Integration of the model with Unity:**

In the study in their study for Gesture Recognition for non RC Drone have successfully imported Leap Motion SDK and Gesture simple control package and used it for detection.

This shows that any kind of trained model can be imported into Unity.

Thus here is a recommended method to import ML model into Unity:

**STEP 1:** Save and export the model to ONNX format as this is the format supported by Unity.

**STEP 2:** Import Unity ML Agents

**STEP 3:** Import the model into Unity.

**STEP 4:** Access the camera feed frame by frame.

**STEP 5:** Resize the render texture if required.

**STEP 6:** Read the Softmax layer output from the model.

**NOTE:** The last two steps are required to be done in the C# script. But before that it is required to properly setup the AR Camera and the environment, for which it is required to select an AR SDK.
Selection of AR SDK:

SDK has multiple components within the application: identification, tracking, etc.

There are many AR SDKs to choose from like: Vuforia, Metatio, Wikitude, ARToolKit, D’Fusion, ARmedi, ARCore, ARKit.

There is another study by Anasse HANAFI, Lotfi ELAACHAK and Mohammed BOUHORMA in which they are focusing on those SDKs which give a platform and support and function with hardware.

They had tested these SDKs on the basis of licence type, target platform, development platform, tracking type, functionality.

The most important feature as per our requirement is tracking type and the results are tabulated in Table 2.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Image Target</th>
<th>Ground &amp; plane detection</th>
<th>Motion Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCore</td>
<td>Not Stable</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>ARKit</td>
<td>Not Stable</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>Vuforia</td>
<td>Stable</td>
<td>Stable</td>
<td>Not Stable</td>
</tr>
</tbody>
</table>

Table 2: AR SDKs Comparison based on Tracking

Before deciding we would like to see specific implementations of each SDK:

Vuforia:

Gagan Kishor Upadhyay, Divij Aggarwal,


Their observation is shown in Table 3:

<table>
<thead>
<tr>
<th>Distances in centimeters</th>
<th>Time Detection</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 cm</td>
<td>0</td>
<td>Product not detected</td>
</tr>
<tr>
<td>30 cm</td>
<td>1.5</td>
<td>Product detected after long time and rendering objects are not perfect</td>
</tr>
<tr>
<td>25 cm</td>
<td>1.5</td>
<td>Product detected after long time but the result is perfect</td>
</tr>
<tr>
<td>15 cm</td>
<td>1</td>
<td>Product detected quickly and the result is perfect rendering</td>
</tr>
</tbody>
</table>

Table 3: Vuforia Results based on distance

Thus we can see that the results are perfect for our scenario, so it can be considered to use Vuforia but from Table 2 our requirement is of Motion Tracking thus it would be a more suitable choice would be to go for ARCore or ARKit.

ARCore and ARKit:

ARCore is used to build for Android Devices and ARKit is used to build for iOS devices. Unity offers a wrapper class known as ARFoundation which includes both of these SDKs and an added advantage is that it is not required to import these from an external source.
Zainab Oufqir, Abdellatif EL ABDERRAHMANI and Khalid Satori did a study to discover if ARFoundation is a suitable replacement for ARKit or ARCore. Their observations are shown in Table 4.

<table>
<thead>
<tr>
<th>Supported Feature</th>
<th>ARFoundation</th>
<th>Unity ARKit for Unity</th>
<th>Unity ARCore for Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face Detection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Plane Detection (Vertical)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Plane Detection (Horizontal)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Feature Point Detection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Light Estimation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HI/Texting (Feature point and Plane point)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Image Tracking</td>
<td>In development</td>
<td>✓ (Static Only)</td>
<td>✓</td>
</tr>
<tr>
<td>3D Object Tracking</td>
<td>In development</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Environment Probe</td>
<td>In development</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hand Mask</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pose Tracking (Face, Head, Hand)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cloud Anchor</td>
<td>In development</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4: ARFoundation features

As it can be seen from Table 4 that ARFoundation either has all the features or they are under development. Thus to keep the implementation simple, future ready and cover all target platforms it is suitable to use ARFoundation as our SDK.

ARFoundation can be imported from Unity’s Package Manager which comes along with Unity during installation.

III. CONCLUSION

In this paper, we had a look at various computer vision techniques for implementing marker less tracking on hands and also determined an appropriate game engine to import that model into. We also compared various Augmented reality SDKs which can be used to properly instantiate the desired objects.

IV. REFERENCES

8. Brent Cowan and Bill Kapralos “A Survey of Frameworks and Game Engines for Serious Game Development” in 2014 IEEE 14th International Conference on Advanced Learning Technologies