Sunny Miglani

Computer Vision Markers And 3D Distance Tracking

Introduction

This technote shall introduce the concept of "Markers" used in Augmented Reality and Computer Vision algorithms, with a focus on the implementation of these Markers in the Puzzle The City application by Anthromos.
Included in this tech note is the solution to a problem faced by Anthromos in trying to calculate the distance between two Markers, as needed for the game logic by the application.

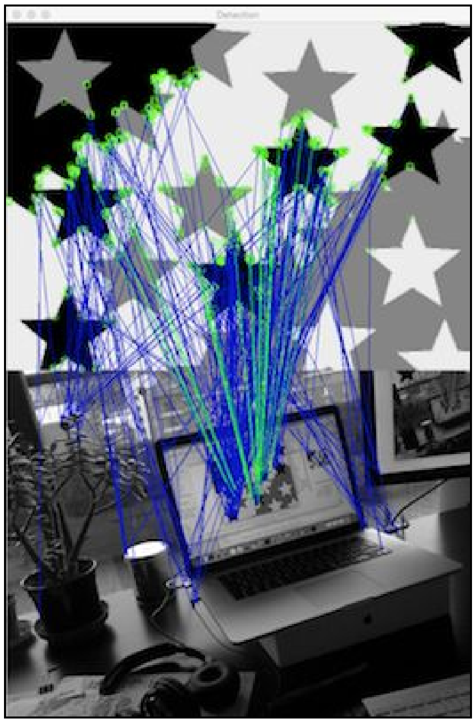
What is a Marker?

A Marker is a high contrast image, made up of sharp edges and distinct lines, that can be used as a unique identification point in a computer vision algorithm. The use of marker based computer vision allows for easy detection of structures and objects, and hence can be used to keep track of any moving objects in the camera feed.
The markers are essentially tracking points for any basic component of a computer vision algorithm as they are robust and reduce computational requirements [1]
Markers act as tracking points for a computer vision algorithm as they provide a robust focus point for the algorithm's tracking feature, reducing the computational requirements for the program. This, in comparison to markerless computer vision, is the better option for use of Augmented Reality on a smartphone device as it reduces the amount of resources needed to run the application.

Use of a Marker's 3D Position Tracking:

The need for tracking the 3D position of the markers is to allow the Puzzle The City's game logic to check how close each physical marker is to its neighbours where the distance and angle would decide whether a pair of pieces are "connected".
During the construction of this app, a lot of time was spent understanding and interpreting Kudan's API documentation, and the solution mentioned in the report seems to be the most optimal method of calculating distance (in this version of the API)
Below is some information about Markers and the different types attempted while selecting markers for the PTC application.

Main Characteristics of a Marker



The design of a marker can get quite complicated based on the use case of the Computer Vision associated with it. This section will focus on the key aspects of markers needed for Puzzle The City.
The design of the marker for PTC requires the following key characteristics [2]

- High Contrast
- Good Detail
- Unique with respect to the other markers
- Sharp Edges

An example of this can be seen in the image alongside, where each part of the image is under high contrast and has key points that can be recognised quickly by a computer. This pattern also stands out from the background and therefore prevents any confusion between the background and the markers themselves

The main focus of this design guide for Markers is based on the fact that having markers too similar or too detailed can cause problems. Below are some attempts at markers for Kudan's AR API, it shall focus on the second and third points of the Marker Design criteria

Puzzle The City Marker Attempts

Step 1: QR Codes :

The initial markers in the PTC app were the intuitive choice of using QR Codes as they provide the high contrast and uniqueness needed for the computer vision algorithm created by Kudan. The markers were based on random words chosen that showed a good amount of uniqueness between the various shapes. Attempting QR codes made it very obvious that algorithms designed for noticing the subtleties of QR codes may not be the same as the algorithms used for computer vision tracking.
Problems arose with keeping track of multiple similar markers and using them in various backgrounds and light conditions especially with the intricate details of a QR code in comparison to the example image given above.
The QR codes' minute details caused problems when in noisy background as the detail in QR codes was too fine for the camera to find it in a noisy background.

Step 2: Human Designed Markers

A quick fix for the small detail markers created by simple QR codes was to use human made markers, i.e. markers designed in

photoshop or paint trying to fit towards the given requirements of high contrast and good detail in all images without having the details be too small.
The problem with designing the markers ourselves, was that the markers started having a similar overall structure. This lead to similar problems with the QR codes and user friendliness.

Out of the 16 proposed markers, some designs took too long for the algorithm to recognise and there were problems in tracking during movement of these markers which heavily disrupted the immersion of the user into the app's environment. With such a small set of markers already creating problems, it raised concerns on the scalability of human designed markers.

Step 3: Pebble Marker Designs

A suggested solution by Kudan's team was to use an image of pebbles split up as the image consisted of enough randomness that it would not cause problems for the CV algorithms to recognise and the structures of pebbles would not have any similarity to common day backgrounds. While testing these markers, a large improvement was seen in the time taken to recognise and track these markers.

While these objects may look similar to the human eye, the computer vision algorithms seem to favour the sharp random edges created by the pebbles allowing for better tracking during movement. The low effort needed to create these markers was quite a useful factor as it would allow scalability to any number of markers. It was also noticed that repeating patterns do not work well for the given CV algorithm, and so the randomness offered by pebbles was of good advantage.

The above images show a set of 9 markers made from one image of a pebble beach, and the single zoomed in version of a marker showing the high number of high contrasted edges.

Tracking of Markers in 3D Environment

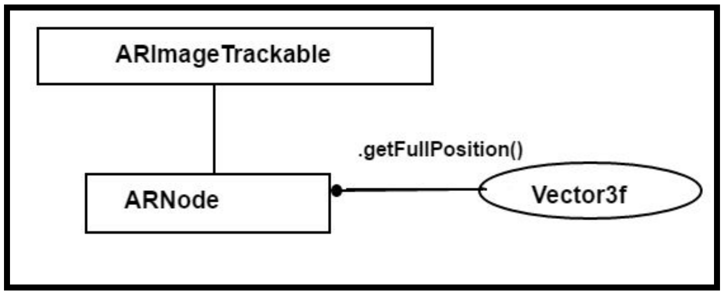
Challenge :

A problem faced when creating the Puzzle The City application was to find the distance between each “puzzle piece” and check whether the objects were close enough to be considered “connected”. This pieces in this instance were physical “Markers” being moved by the user.

With the limited description available in the Kudan API’s documentation, it was hard to find the most optimal way to calculate distance while keeping the code clean and the running time of the algorithm fast.

Environment:

Each marker is stored as a child or “ARNode” of an “ARImageTrackable” which keeps track of the marker in the Camera stream of the application.
The position of the markers is stored in a Java Class called *“Vector3f”* which keeps track of the positions as a 3 element vector that is represented by single precision floating point x,y,z coordinates.



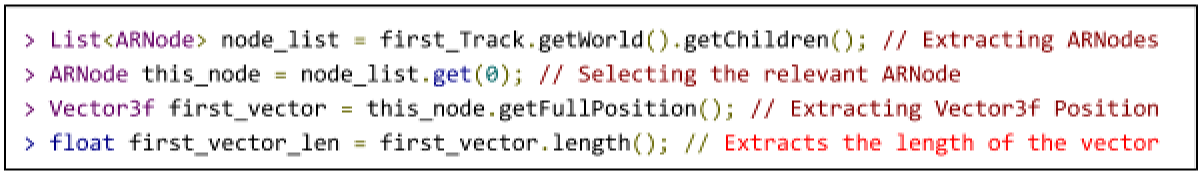
To obtain this Vector3f value, we use the ARNode’s “.getFullPosition()” function which returns the position of the object relative to it’s parent (the “ArImageTrackable”).

This structure is duplicated for each of the 16 possible markers in the set of Markers (or ARImageTrackables)

Solution:

Using the data extracted from the Vector3f object, finding the distance between any two markers becomes quite straightforward with the following methods.

The steps are outlined in the code snippet below

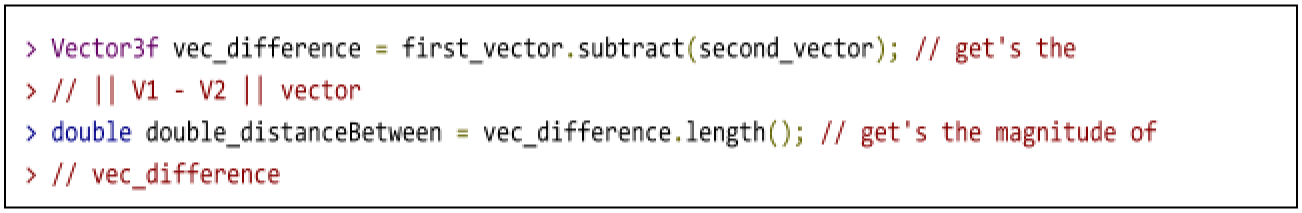


```
> List<ARNode> node_list = first_Track.getWorld().getChildren(); // Extracting ARNodes
> ARNode this_node = node_list.get(0); // Selecting the relevant ARNode
> Vector3f first_vector = this_node.getFullPosition(); // Extracting Vector3f Position
> float first_vector_len = first_vector.length(); // Extracts the length of the vector
```

The value of *“first_vector_len”* returns the distance of the marker to the camera.

The best way to find the distance between the two pieces is by extracting the Vector3f of the two markers, apply vector subtraction to find the vector set that’s the middle of the pieces.

The vector distance of this point is then found as so :



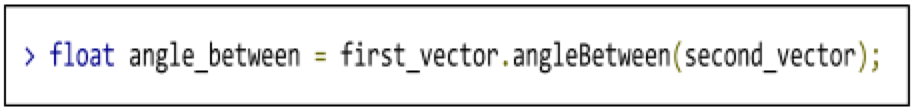
```
> Vector3f vec_difference = first_vector.subtract(second_vector); // get's the
> // || V1 - V2 || vector
> double double_distanceBetween = vec_difference.length(); // get's the magnitude of
> // vec_difference
```

Where “*double_distanceBetween”* is the variable containing the length between the two relevant pieces.

Application to Game Logic:

Applying this data to game logic is done by iterative testing, trying different distance values to be able to find the perfect distance that counts as “connected” to each jigsaw piece. This distance works along with the idea of using the angle between the two marker’s Vector3f values.

To calculate the angle is done using the Vector3f’s “*.angleBetween(...)*” function



```
> float angle_between = first_vector.angleBetween(second_vector);
```

The game logic of Puzzle The City runs on the conditions that each piece satisfies the inequalities of being less than the distance and angle required for a piece to be considered as “connected”.

Discussion

The aim of this technote was to introduce Markers and the optimal design for them. Along with this, introduce a solution to a fairly complex problem with a simple solution.
It is important to note that the solution may seem straightforward now, but is a large task when faced with limited documentation.
This solution should be a guide to a fairly common need in using Kudan’s software and API and should hopefully solve future problems for users of the API.

References :

1) “Marker-less Tracking for AR: A Learning-Based Approach” Link : http://www.riedelcastro.org/publications/papers/genc02marker-less.pdf
2) “What makes a good marker?” Link: https://wiki.kudan.eu/What_Makes_a_Good_Marker%3F