BAHÇEŞEHİR UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

VR GLASSES FOR VIRTUAL REALITY APPLICATION

Submitted by

Omran Kaddah. (STUDENT NAME)

1337525….. (STUDENT NO)

Advisor

Yrd. Doç. Dr.

Tarkan AYDIN

İSTANBUL, May 2017

BAHÇEŞEHİR UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

This project work submitted by **Omran Kaddah** has been done under my supervision. I hereby state that the work outlined in this report satisfies the requirements of the compulsory "Capstone Project" course, and **Omran Kaddah** can take the capstone project examination.

Signature and Date

…………………………

ADVISOR NAME-SURNAME

Omran Kaddah has taken and passed the capstone project examination in our presence on ……., ……., 2017. We hereby certify that this project work fulfills all requirements of "Capstone Project" course.

THE EXAMINATION COMMITTEE

Committee Member	Signature
1. …………………………..	………………………..
2. …………………………..	………………………..
3. …………………………..	………………………..

I have reviewed this capstone project report submitted by **Omran Kaddah** I hereby endorse the project work described here as a "Capstone Project I."

SUMMARY

The project revolves around building a headset similar to contemporary virtual reality headset like Oculus but with budget less than 50$, simplest materials, and tools. This would give the opportunity for people who can't afford buying virtual reality headset, that its position in space is tracked in addition to its rotation, to have the full virtual reality experience. User is only required to have his own mobile device, personal computer, VR headset mount for smart device with Wifi connection, camera, and some model can be made by hand to engage in the virtual world. The design problem is addressed by continuously solving Perspective-n-Points problem of points from distinctive drawn markers attached on the headset, which are easy to be detected and each one of them has its own identification number so the n points are identified. Solving this problems means having an estimate of the 3 rd dimension rotation and translation of a 3 rd object , which is the headset in this case, from a single 2D photo. Then position and rotation of the headset are continuously sent to the mobile device to be processed by virtual reality application. The tracking system of the headset is obviously not robust as other expensive VR headset systems. However everything can be improved in the future to have a robust system.

ACKNOWLEDGEMENTS

I want to thank my Design Project advisor Tarkan Aydın on helping and guiding me on my project and every single person who helped me on creators.xna.com forums. Also I want to thank to my family and friends who have always supported me during this thesis study.

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

Omran Kaddah

İstanbul, May 2017

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ABBREVIATIONS

VR: Virtual Reality

CV: Computer Vision

2D: 2rd Dimension

3D: 3rd Dimension

6DoF: Six Degrees of Freedom

PnP: Perspective n Points

API: Application Programming Interface

IDE: Integrated Development Environment

IMU: Inertial Measurement Unit

ATH : Adaptive Threshold

1 Introduction

Building a functional VR glasses requires, designing glasses that can be tracked, and constructing and programming an application that includes a tracking system for the glasses. However, in this project a commercial glasses are used, and instead a model with ArUco markers attached on it is designed to be mounted on the headset. Regarding the tracking system, which is about 3D pose estimation of the mounted model by solving PnP after the detection of ArUco markers, then finally send the resulting information to the mobile device.

The reason behind working on this project is, to learn about CV and technically apply methods I learn like, pattern detection, pose estimation, and many things in between. Also, to grant a wider group of people the full VR immersive experience.
There has been a lot of research on VR technologies but most of these were targeting to specific users who could afford more than 400$ headset. In contrast, this project will target everyone, and it's the first project to provide 6DoF, that is rotation and position tracking, with a lesser cost. However, there has been a lot of research on estimating a pose of an object. Also ArUco markers are a contribution of Rafael Muñoz and Sergio Garrido et all[1] as you will see in the rest of report, these markers play essential role in the tracking system.
The developed system has shown its ability to track the marked model, fetching the translation and rotation vector and send them to the smartphone. However, due to the limitation in time to design a better model with marker behind the headset, the model is only tracked if it faces the camera. Another limitation is, when the model suddenly move quickly to another position, then then the system lose track of the model for a little time.

2 Related Works

In this section, you must give the literature studies about your project topic in detail. You can also divide this section into further subsections to better organize your project.

In this section, I will discuss literature studies that is related to the project. This section is divided into subsections, first section will discuss Oculus rift VR headset, the rest will discuss the studies related to each step taken to develop the tracking system.

1.1 Tracking systems:

The idea behind the tracking system of the project is inspired by Oculus constellation system. Oculus rift tracking system works as follows. First, the camera gets each IR leds blinking pattern so it knows their IDs after detecting them. Second, with the help of IDs it knows the location of each of the leds according to the coordinate of the headset. Finally, It matches each of leds coordinates in 3D world to 2D of image plane and computes Perspective-n-Points problem, and thus 3D pose estimation of headset. Important aspect of constellation is that IMU-based tracking algorithm date is combined with PnP to get a robust tracking, especially when the headset shift quickly from one position to another, and when it tries to have the full orientation from view visible points.

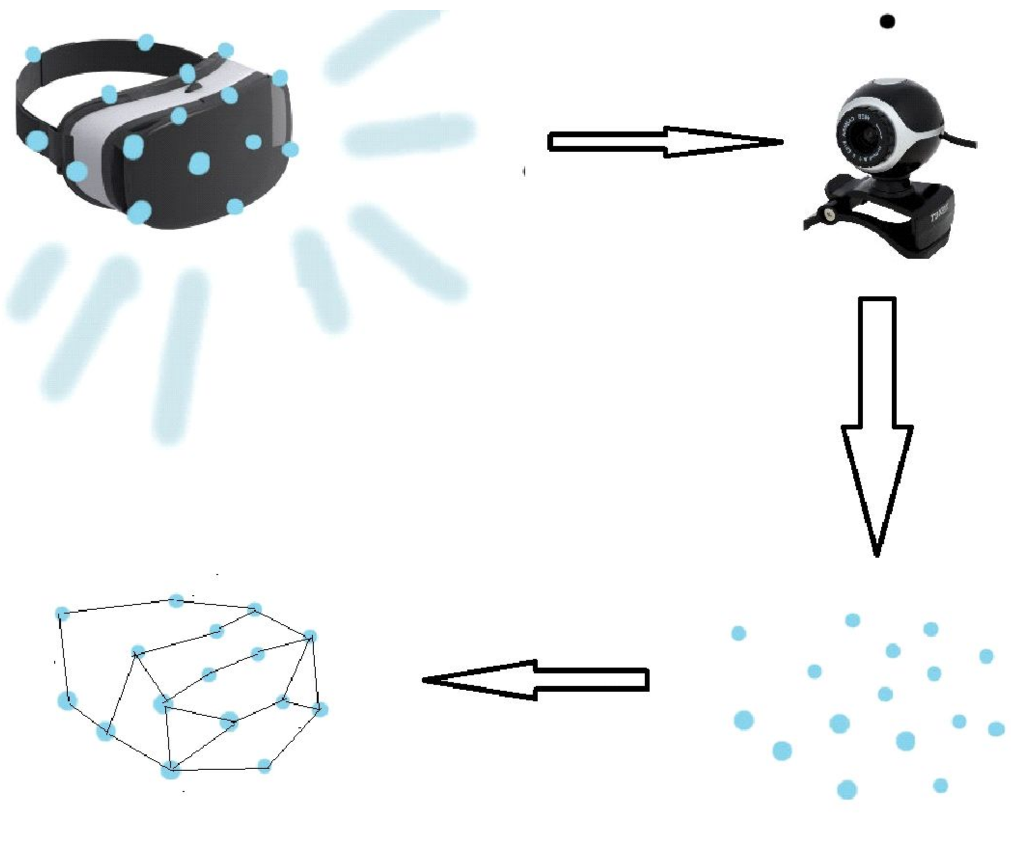


figure 1: Diagram of the tracking system of the Oculus rift

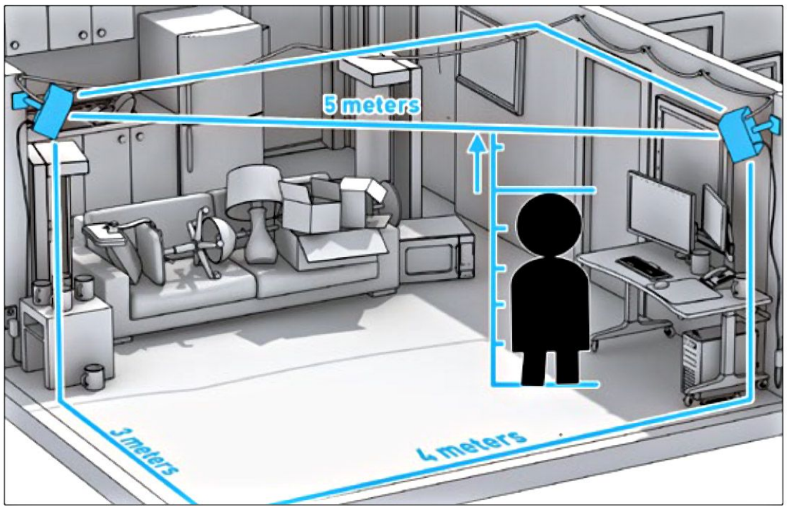
Other Tracking systems for VR headset like “Light House”, which is used by HTC Vive. How the system works can be inferred from its name, where there are two box called Lighthouse Box that are installed on a height above the headset beam IR lights signals at 60 Hz which then the vive's IR-diodes can see. Those sensors dotted all over the Vive pick the light of the LEDs, and the headset begins timing the millisecond it gets hit by this light. Then, it waits until it gets hit by one of the lasers, and uses the data it collects to determine *where* the photosensor that was hit is, and *when* the beam was in contact with the headset. That way, the headset can instantly calculate its *exact position* relative to the base stations.[2] Advantages of Light-house system over Constellation can been seen with the lighthouse boxes:

- they are passive. Their components are simple and cheap.

-they don't need a high bandwidth connection to any of the VR system's components (headset or pc).

-tracking resolution is not limited or narrowed down to the camera resolution like on conventional solutions.

and with resources consumption, the system is efficient because position/orientation calculations are fast and easy handable by (more) simple CPUs/microcontrollers. No image processing cpu time is consumed like on camera based solutions.



1.2 ArUco Markers:

ArUco markers are rectangle shape image, containing black and white areas inside in a code-like pattern. The black and white area is divided into equal sized cells, in which each cell contain one color, and the border or the frame of the marker is black and has a thickness of one cell.

Note: In this project, a square shape ArUco Markers are used.

As shown in the Figure.3, which is an example of ArUco marker. This marker is 4x4 marker because it has 4 cells in each row. First assumption would be, there would be $2^{4*4} = 65536$ possible markers. However, this is not the case in ArUco markers, the codification employed is a light modification of the hamming code.[3]

So in this example the word is 4 bits long, thus the first, and second bit is used for error detection. This implies the marker 4x4 has 2^{2*4} possible code or ID.

The main difference with the hamming code is that the first bit is inverted. So when ID: 0 is 0000 becomes 1000 this will prevent a completely black square from being a valid marker ID[3] The goal of do that is to reduce the possibility of detecting objects that rectangular and are completely black.

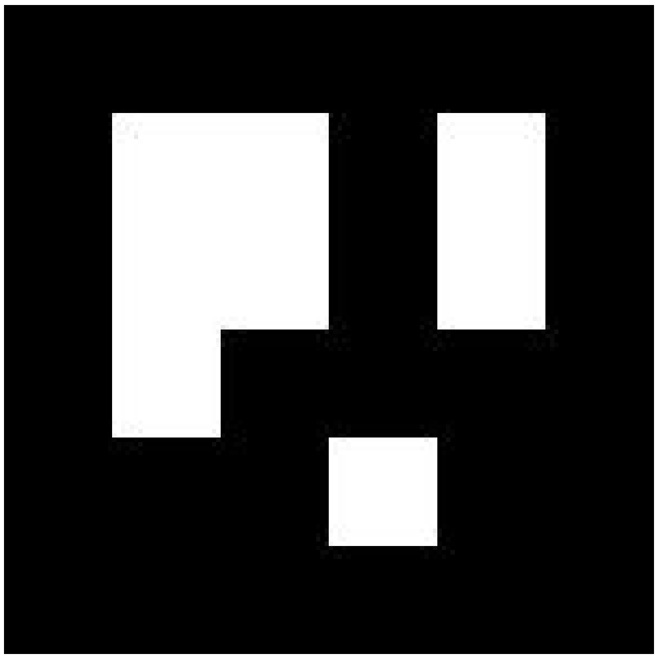
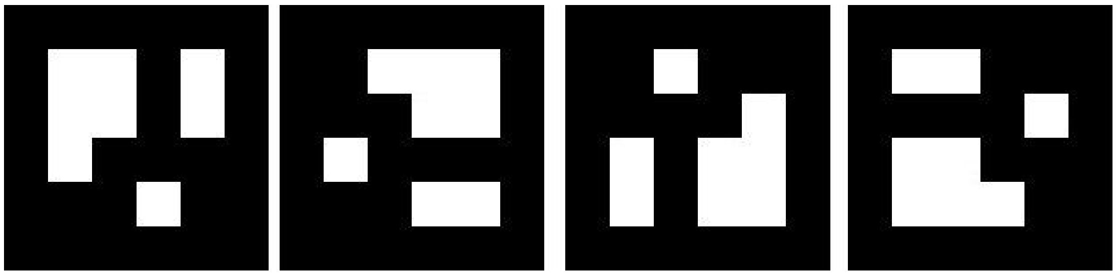


Figure3 An example of an ArUco 4x4 marker.

Here is an example where the first marker has the correct orientation, has hamming distance of Zero, while others have bigger than zero.This will help finding the correct orientation of the marker.

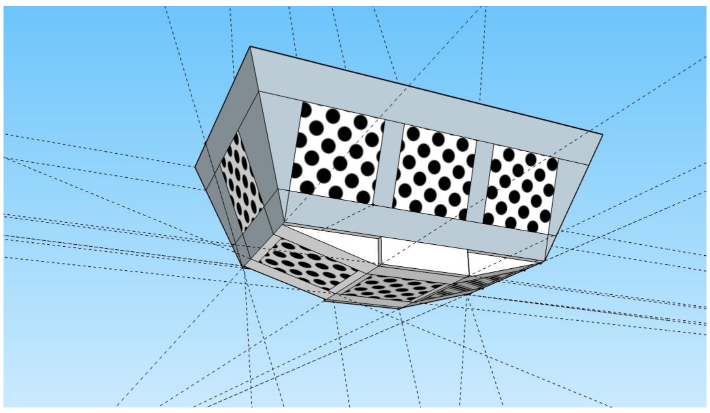


3 Materials and Methods

This section will give a detailed account of steps were taken to complete the project. First section will discuss the headset and the model that is mounted on it. The rest of the sections will give an elaborated explanation about the methods and algorithms implemented in the tracking system, and they will follow the order of each procedure in the program.

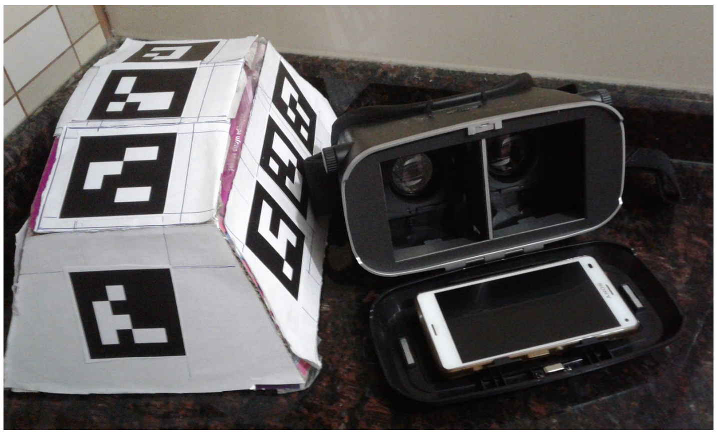
3.1 VR headset, Mounted model, and other hardware parts:

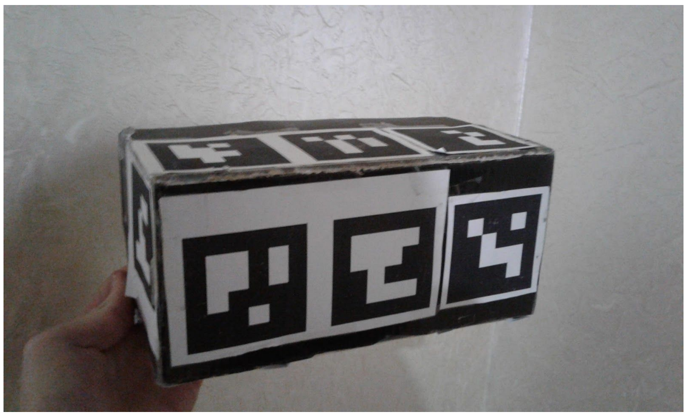
In this project, a commercial VR headset that requires a smart device to function, Frustum-shaped model is mounted on the. The model was designed to have as much plane as possible to have a better results from PnP problem and to use some methods which requires that point to be on non-planar plane.



Model on SketchUp where all the measurement were done it.

The picture on the left shows the model on the left and commercial VR headset on the right.The other picture shows how the model is mounted on the VR headset.





The older version of the Mounted Model. Was discarded because it has only 5 planes, and did not show a lot of marker at the same time.

3.2 Introduction to the Tracking system:

Now we proceed logical part which is the tracking system. The next coming subsections will discuss solely each of the phase that the program take to track the headset in each frame, how it is implemented, and the theory behind it. Regarding this section, it will give a general and broad idea about how the whole system works.

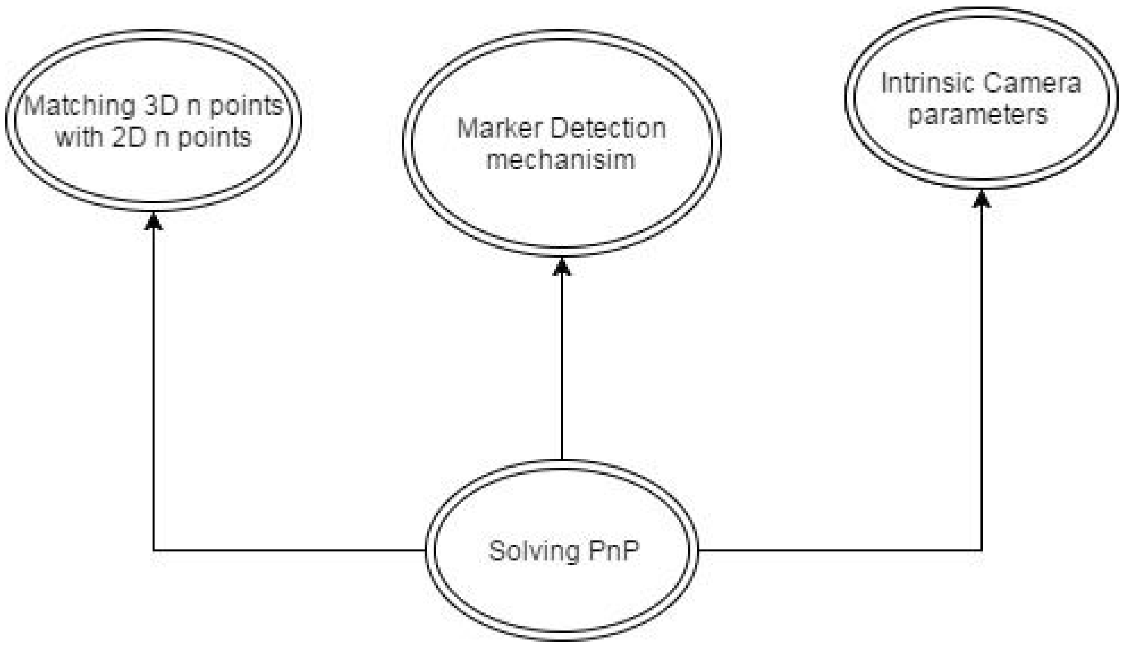
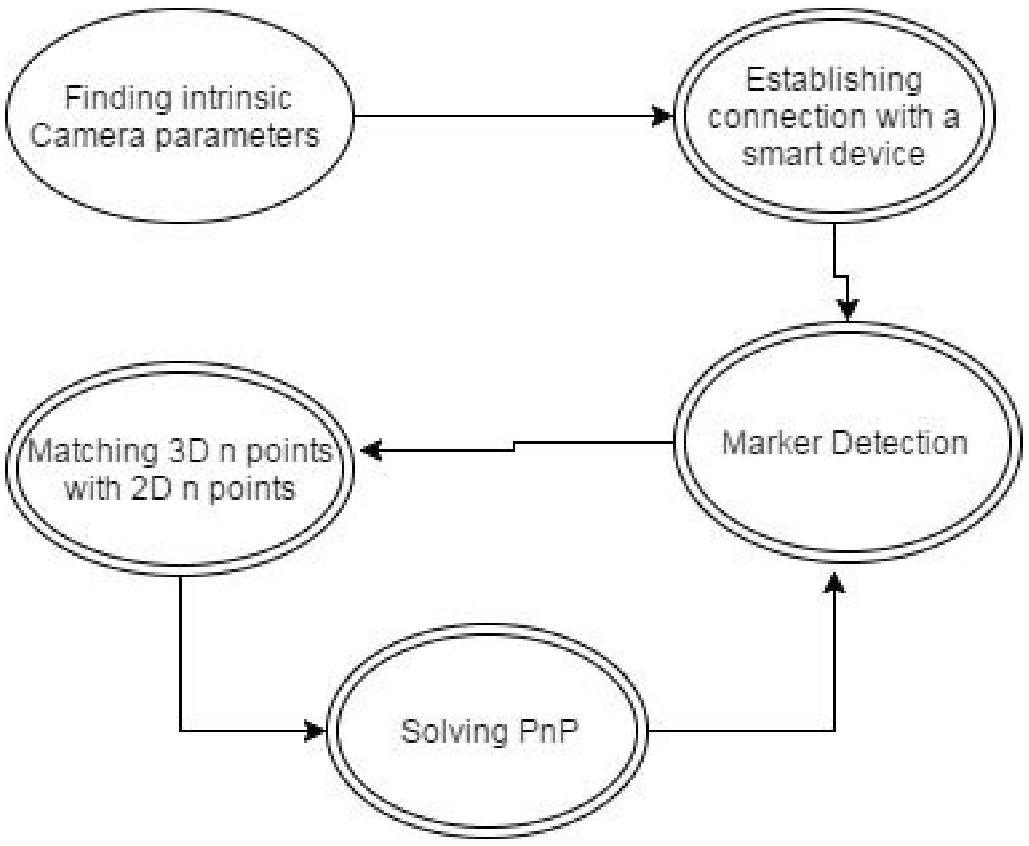

Matching 3D n points with 2D n points
Marker Detection mechanisim
Intrinsic Camera parameters
Solving PnP

Figure 4 Dependencies of PnP problem on other Components in the program

The tracking system is based on solving PnP problem to get the 6DoF, translation and rotation vectors, of the headset according to the camera. However, PnP requires 2D projections of the 3D points in real world. So 2D points have to be found and matched with corresponding 3D points. This done by finding marker corners of the ArUco markers . With the help of marker's ID number, the detected marker's position and place according to headset's mounted-model's coordinate system will be know, otherwise program won't understand which marker is detected. After that, 2D projections of the 3D points of the markers corners should be matched together to solve PnP. We shall not forget that, PnP can't be solved without having the intrinsic camera parameters, thus the program should check whether there parameters are available or not. If not, then camera should be calibrated. This implies, getting the camera intrinsic parameters shall be the first phase in the program. Followed by finding translation & rotation vectors streaming to smart device's mechanism as second step, which is going establishing a TCP-IP server and waiting for a connection with a client. Then the tracking system incept, searching and detecting the markers, matching markers corners in the image plane with corresponding real 3D world, solving PnP, sending the results to smart device, going back to the starting point, and so on until the halting the program.

Figure # Components of the program that make the tracking application and their sequence


Finding intrinsic Camera parameters
Establishing connection with a smart device
Marker Detection
Matching 3D n points with 2D n points
Solving PnP

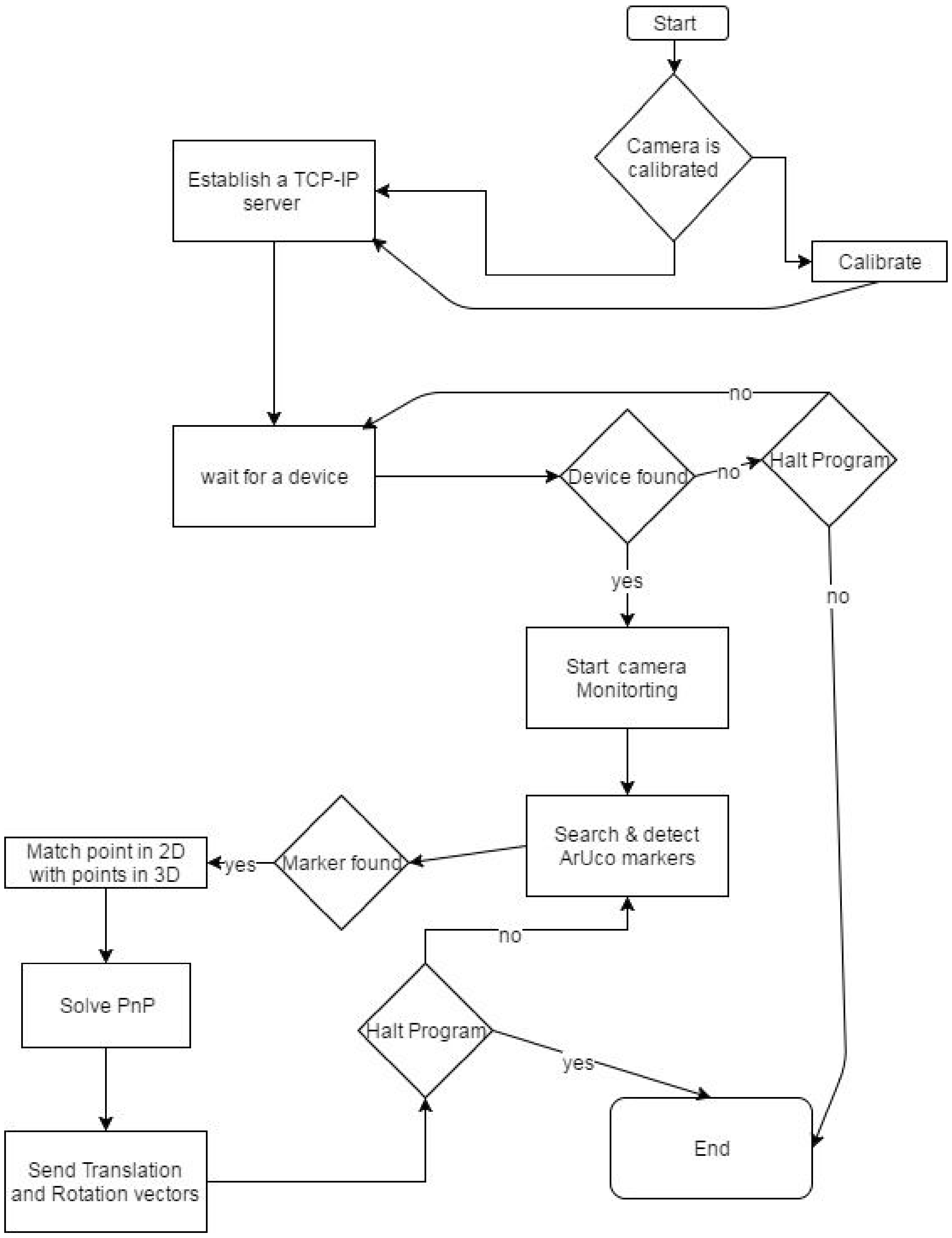


Figure This flow chart diagram depicts the implemented code in the main function of the program.

3.3 Camera Calibration Phase:

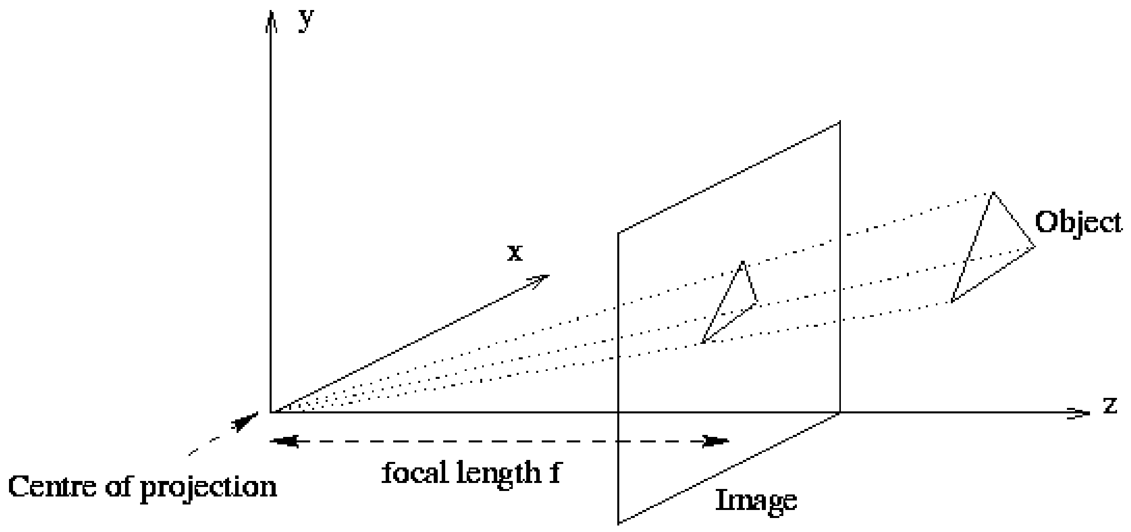


Figure # Illustration of the camera model

Camera calibration is the process of estimating the camera's linear intrinsic parameter, which is know as camera matrix; and camera's non-linear intrinsic parameters, which is also known as distortion coefficients of lens.

What are the intrinsic parameters?

$$s\,m' = A[R|t]M'$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(X, Y, Z) are the coordinates of a 3D point in the world coordinate space

(u, v) are the coordinates of the projection point in pixels or projections at the image plane

A is a camera matrix, or a matrix of intrinsic parameters

(cx, cy) is a principal point that is usually at the image center. Def. principal point is the image point lying on the principal axis of the camera. It is the point in the image that is closest to the center of projection. Correspondingly, the principal axis (or optical axis) of the camera is the line through the center of projection normal to the image plane.[#1]

fx, fy are the focal lengths expressed in pixel units. Def. Focal length the distance between the centre of a lens or curved mirror and its focus. Def. Focal point is where the rays of light after reflection or refraction.

The is also the skew but the algorithm that is used in this project doesn't compute it

What are the distortion coefficients?

They are coefficients related to how deformed and bended the lens is. They also referred as to Optical Distortion.[4] Due to the fact the pinhole camera does not have a lens, distortion coefficient are not included in the camera Matrix.[5]

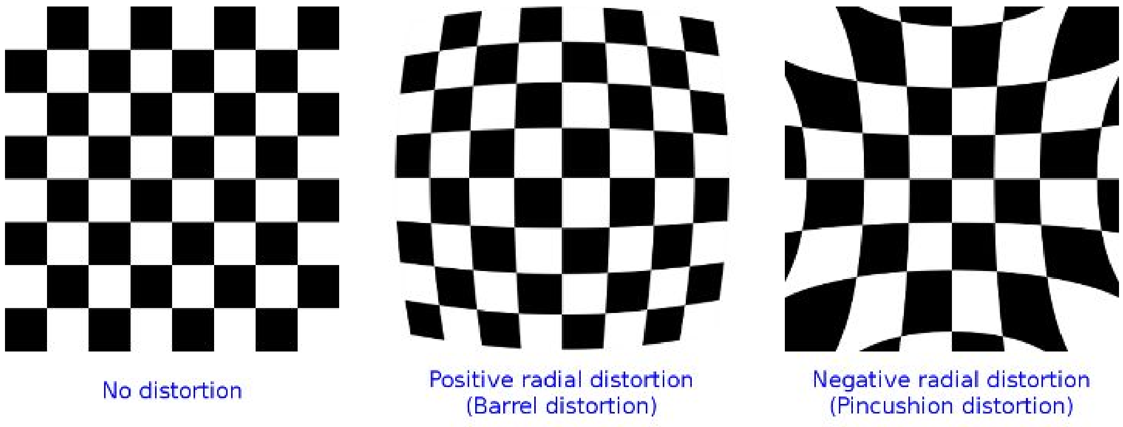
TH

Radial Distortion

Radial distortion occurs when light rays bend more near the edges of a lens than they do at its optical center. The smaller the lens, the greater the distortion.[5]The radial distortion coefficients model this type of distortion. The distorted points are denoted as ($x$distorted, $y$distorted):

$$x\text{distorted} = x(1 + k1{*}r2 + k2{*}r4 + k3{*}r6)$$

$$y\text{distorted} = y(1 + k1{*}r2 + k2{*}r4 + k3{*}r6)$$


No distortion
Positive radial distortion (Barrel distortion)
Negative radial distortion (Pincushion distortion)

x, *y* — Undistorted pixel locations. *x* and *y* are in normalized image coordinates. Normalized image coordinates are calculated from pixel coordinates by translating to the optical center and dividing by the focal length in pixels. Thus, *x* and *y* are dimensionless.

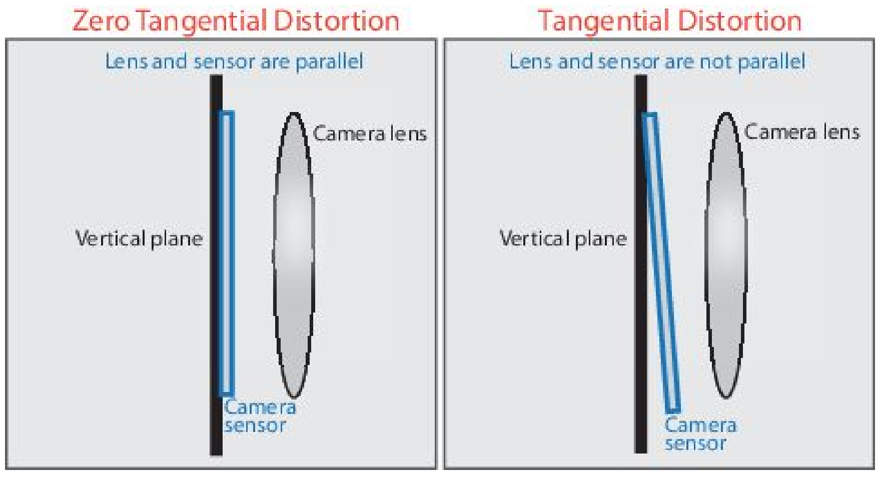
*k*1, *k*2, and *k*3 — Radial distortion coefficients of the lens.

*r*2: *x*2 + *y*2

Typically, two coefficients are sufficient for calibration. For severe distortion, such as in wide-angle lenses, you can select 3 coefficients to include *k*3.

Tangential Distortion

Tangential distortion occurs when the lens and the image plane are not parallel. The tangential distortion coefficients model this type of distortion.



The distorted points are denoted as ($x_{distorted}$, $y_{distorted}$):

$x_{distorted} = x + [2 * p_1 * x * y + p_2 * (r_2 + 2 * x_2)]$

$y_{distorted} = y + [p_1 * (r_2 + 2 * y_2) + 2 * p_2 * x * y]$

x, *y* — Undistorted pixel locations. *x* and *y* are in normalized image coordinates. Normalized image coordinates are calculated from pixel coordinates by translating to the optical center and dividing by the focal length in pixels. Thus, *x* and *y* are dimensionless.

*p*1 and *p*2 — Tangential distortion coefficients of the lens.

*r*2 = *x*2 + *y*2 [5]

The algorithm that is used to estimate camera matrix and Distortion coefficient

is Zhang method please refer to [5] for more information

Code Side:

This function gets the points/ corners that are common between two black chessboard cells and other two white cells.

```
void PostionOfcornersOnTheBoard(float LenghtOFSquaresEdge, cv::Size BoardSize, std::vector<cv::Point3f> & corners)
```

This function Detect those points in image plane.

```
voidgetChessBoardCorners(std::vector<cv::Mat>imgs, std::vector<std::vector<cv::Point2f>> & FoundCorners, bool showResults = false)
```

This function includes the two previous function and fills camera matrix and distortion coefficients.

```
void cameraCalibration(std::vector<cv::Mat> calibrationImages, cv::Size boardSize, float LenghtOFSquaresEdge, cv::Mat & cameraMat, cv::Mat & distanceCoieffcients).
```

1.3 Markers Detection phase:

Since the PnP problem requires headset's corresponding 2D points in the image plane of the world 3D points, the system has to detect these points and this is what the Markers Detection phase is about. In this section I am going to discuss the detecMarkers(..) function in the source code, which represents this phase.

```
void  detectMarkers1(cv::InputArray  _image,  const  cv::Ptr<cv::aruco::Dictionary>
&_dictionary,   cv::OutputArrayOfArrays   _corners,   cv::OutputArray   _ids,   const
cv::Ptr<DetectorParameters1> &_params, cv::OutputArrayOfArrays _rejectedImgPoints)
{
  CV_Assert(!_image.empty());

  cv::Mat grey;

 _convertToGrey1(_image.getMat(), grey);

 /// STEP 1: Detect marker candidates
 vector< vector< cv::Point2f > > candidates;
 vector< vector< cv::Point > > contours;
 vector< int > ids;
 _detectCandidates1(grey, candidates, contours, _params);

 /// STEP 2: Check candidate codification (identify markers)
  _identifyCandidates1(grey,  candidates,  contours,  _dictionary,  candidates,  ids,
_params, _rejectedImgPoints);

 /// STEP 3: Filter detected markers;
 _filterDetectedMarkers1(candidates, ids);

 // copy to output arrays
 _copyVector2Output1(candidates, _corners);
  cv::Mat(ids).copyTo(_ids);

 /// STEP 4: Corner refinement
 if (_params->doCornerRefinement) {
                       CV_Assert(_params->cornerRefinementWinSize    >    0    &&
_params->cornerRefinementMaxIterations > 0 &&
     _params->cornerRefinementMinAccuracy > 0);
      parallel_for_(cv::Range(0, _corners.cols()),
       MarkerSubpixelParallel(&grey, _corners, _params));
  }
}
```

Code snippet detectMarkers() function

This function takes 6 parameters, two of which are optional. The first parameter is of Matrix type, which is the image frame where the markers to be detected. The second parameter is of Dictionary type, a class to represent a dictionary of markers, explained in the first section. The third parameter, is an array of an array of corners to be filled. After filling process, each individual index of the outer array refers to one marker's array of corners, the corners will be stored in clockwise direction. The fourth parameter is array to be filled, after filling process array will contain, in the same order of previous parameter, a list of detected markers' ID. The fifth parameter is optional, if it was not passed, the default setting of `Detector Parameters` will be used. Detector Parameters are used to customize the processes that take place in detectMarkers, so it will be discussed soon with each process. Final parameter is an optional parameter, an array of an array of rejected points, and can be used if the detection refinement is needed.

Before diving into the implementation of `detectMarkers`, the explanation on how each function in detectMarkers won't be explained in detail because this will require me a lot of explanation, more than 100 pages; detectMarkers is equal to more than 750 lines of codes. However, we go through the main processes and methods applied to detect Aruco Markers.
As shown in this snippet

```
_convertToGrey1(_image.getMat(), grey)
```

This function the first main function to be called. It converts colored image into a grayscale image, where there is only one channel in each pixel, ranging from 0 to 255, the lowest value is black and highest white and gray in between. The conversion to grayscale is necessary because operating on grayscale images is much easier; markers contain only black and white cells so there is no loss in information. After conversion the output is used in the next function `_detectCandidates1(grey, candidates, contours, _params);`

The job of this function is to detect square shapes in the input image, and it consisted of three important function. First function, `_detectInitialCandidates`. This function applies different ranges of Adaptive Threshold to detect contour in each range.

```
((params->adaptiveThreshWinSizeMax - params->adaptiveThreshWinSizeMin) /
params->adaptiveThreshWinSizeStep + 1;)
//default values are used in this project
//this is a snippet from _detectInitialCandidates implementation
```

Here is the first use of DetectorParameters, where the values for the Adaptive thresh window's size can be assigned, since this threshold operation depend on the surrounding pixels to assign the examined pixel a value of zero or one, i.e black and white. The outcome in every range in ATH is black and white image, then within this range a search for contours is applied by _findMArkerContours. In this function, `DetectorParameters` is also used again to assign some constants related to ArUco markers, in this project these constants were not manipulated because the ArUco markers were generated with the default settings too. _findMarkerContours() does these operations to get the required contours, starting with finding contours then it uses approxPolyDP() to approximate to polygon, checking the approximated Curve that was assigned in approxPolyDP() whether it has size of 4 and weather it is a convex, checking minimum distance between corners, checking if contour is too near to the image border, finally if those previous checks and test were all passed the contour is stored in array. Then the rest of functions in `_detectInitialCandidates` reorder contours' points in clockwise direction, and filter the candidates that are too close to each other and remove the smaller one.

Now the contours are passed to second main function in `detecMarkers(),` `_identifyCandidates().` This function checks if these detected squared shapes are

AruCo markers and extract their ID if they happened to be, it also save the rejected candidates if it is required. This function gets rid of perspective distortion by computing perspective transformation first getPerspectiveTransform(),
applies perspective transformation warpPerspective(), then applies first threshold using Otsu's method, and starts reading cells from top left corner.

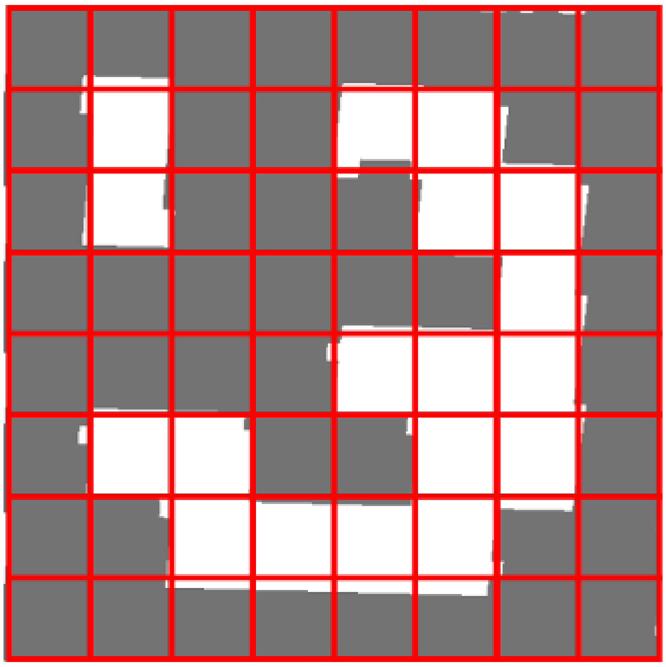


Figure how the marker is virtually segmented, depending on cell size. Ref. AruCo markers documentation.

The bits pattern is checked against the dictionary to determine whether it exist or not, if so, the ID of marker is fetched and the maker is rotated to the correct orientation/rotation of the marker.

Last none optional detection-related function in `detectMarkers()` is
`_filterDetectedMarkers1(candidates, ids);// detected candidates, ID of those candidates`
This function is responsible for doing the final filtering process for candidates, the filtering process. This process only include removing repeated markers with same id, and one if it contains the another.

Now we reached the end of the detection operation, and skipped the last optional function because it is not used in this project. The last function is only responsible for copying the coordinates of detected markers' corners to an array of an array but the implementation was change for this project to one dimensional array so it could be used for solvePnP(). Also copying IDs of the detected markers to an array in the same order corners array. And that concludes the detection section.

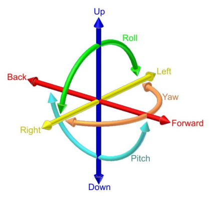
3.4 Solving PnP Problem Phase:

In the previous phases, the tracking system provided all the requirements for solving Perspective-N-points problem. These were, camera matrix, distance coefficients, coordinates of points in image plane(2D). We already have the 3D coordinate of points since they are predetermined and measured in the real world. However, we need only to retrieve the coordinates of detected points, and then push them into a array container(vector<cv::Point3f) in which they are aligned with array container of 2D coordinates (vector<cv::Point2f>) i.e. Index of the first array contains the 3D coordinates of point x then the first index of the second array should contain the 2D coordinates of the same point x.
This is done by the function `object World Coordinates();`
which takes the IDs of detected markers as arguments, and vector<cv::Point3D to be filled.

Finally, we reached the final phase in determining the headset position in space and its orientation in the given frame, solving Perspective-n-Point Problem.
What is PnP problem?
In the section the subsection **3.3** we tried to estimate the intrinsic parameters. What solving PnP does is the estimation of extrinsic camera parameters, [R] and [t] rotation vector (roll, pitch, and yaw) and translation vector (x,y,z), and that is a total of 6DoF. By doing so, solving PnP estimates the pose of the camera to n points. Thus we could know where the object is located and oriented according to the camera.

$$s\ m' = A[R|t]M'$$

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_1\\r_{21} & r_{22} & r_{23} & t_2\\r_{31} & r_{32} & r_{33} & t_3\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$



s m = A R T M

There many different Methods to solvePnP; the selection of the method depends on the application. E.g using Efficient P*n*P (EP*n*P) which is included in OpenCV library is robust (refer to experimental results section) but it does not give an accurate results. In this application, the Iterative method is used to solve PnP. According to the documentation of Opencv, CV_ITERATIVE Iterative method is based on Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections image Points and the projected (using projectPoints()) object Points .

Here is a code snippet from the program of the project:

```
 cv::aruco::detectMarkers(frame,Dictionary, markerCorners,markersID);
 spreadedCornerPoints(markerCorners, spreadedCorners, frame);
 objectWorldsCordinates(markersID, markerEdgeLenght, distancneBEM, realWorld);



if (markersID.size() > 0){

       if (cv::solvePnP(realWorld, spreadedCorners, cameraMat, distanceCoefficient,
rotationVectors, transaltionVectors)) {
                             drawAxis(frame,    cameraMat,    distanceCoefficient,
rotationVectors, transaltionVectors, 0.1f);
                             return 1;
```

```
                    }
            }
```
As we can see, the method was not assigned to the solvePnP because the default method is CV_ITERATIVE which has a value equal to Zero. However, after some experiments EPnP (efficient PnP) , Please refer to experiments section.
First argument is the 3D coordinate of points according to headset coordinate system.
Second argument is the 2D coordinate of points in the image plane

4 Experimental Results

In this section, a demonstration of the main steps taken during the detection process of markers will be shown. IDE used in this experiment is Visual Studio 2017. Also in this experiment, the receiving device was an Andriod. The received information was in this pattern X;Y;Z;roll; pitch;yaw;

Image in the grey scale

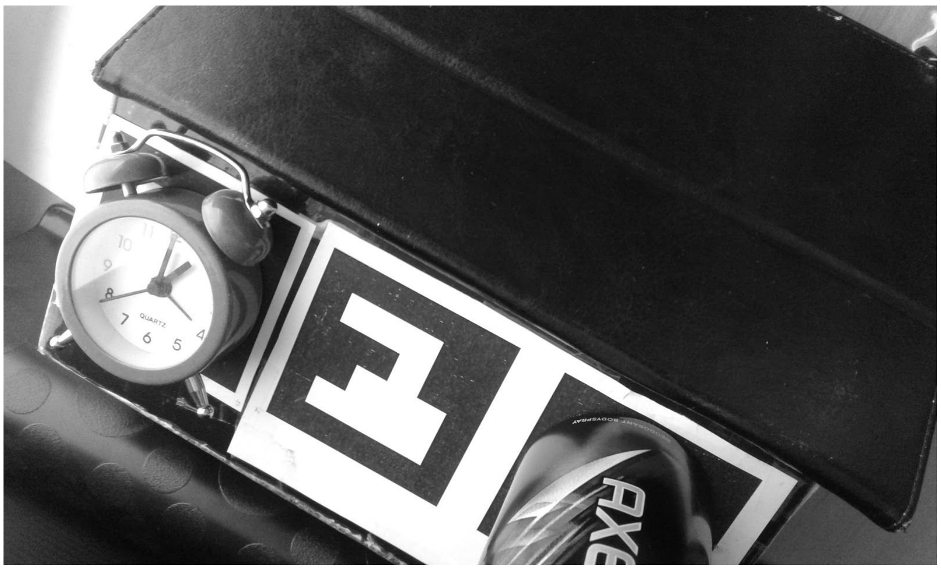
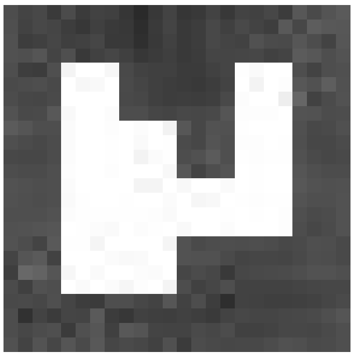


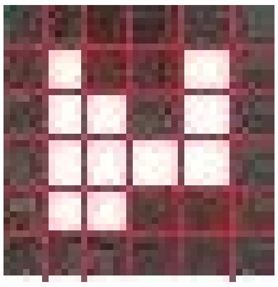
Image after adaptive threshold

Contour detection

The marker after perspective transformation.



After removing perspective projection of marker, the marker is divided into equal sized cells.



Bit Pattern = 1001 1101 1111 1100 .   ID = 24

Also solving PnP results, different experiments have been performed to examine every algorithm.

Experiments when the front markers appear, or when more than 5 markers appear:

Iterative method, was the slowest with 8ms and it did not gave the best results compared to EPnP. The resultant translation and rotation vector came with good results with low error error propagation/ration but when the headset starts to go away from the camera with distant of 1,3 (Avg) it starts to give bad results with high errors. The results get even worse when approaching 2M and if a function like draw axes is used, the program stopped because one of assertions is not met, that is due to unreasonable results.

EPnP, was the fastest with 3ms , and kept giving the low error error propagation/ration in close distance, an acceptable results between 1.3m and 2m. After the headset exceeds 2M the error ratio increases as the distance increase, which lead to unacceptable results, that the draw axes function crachs. EPnP gave the best results due the fact that many of points are not on the same plane.[7] Although every 4 points are on the same plane, the mounted model has a lot of different planes, which makes the model somehow close to model with non-planar surface.

The other algorithms, like Direct Least Square Methods gave results similar to Iterative but was faster with 3ms as average.

Experiments when the front markers do not, or when less than 5 markers appear:

The error ratio was higher for all of the methods, and the results were not acceptable after distance of 1m.

5 Conclusion

The VR headset prototype is successfully tracked by the output of tracking system, rotation and translation vector. Further improvements to the hardware and software would allow for more robust tracking to the headset. And everyone would have chance to get the full immersive VR experience. Improvements may include, combining the results of IMU to support PnP, using more Markers and more planes.

6 References

[1] S. Garrido-Jurado, , R. Muñoz-Salinas , F.J. Madrid-Cuevas , M.J. Marín-Jiménez Computing and Numerical Analysis Department, Córdoba University, Edificio Einstein, 3a planta, Campus de Rabanales, 14071 Córdoba, Spain. Automatic generation and detection of highly reliable fiducial markers under occlusion (June, 2014)

[2] https://www.gadgetdaily.xyz/heres-exactly-how-the-htc-vive-works/

[3] Mastering OpenCV with Practical Computer Vision Projects. Daniel.Baggio et al page 72

[4] https://www.mathworks.com/help/vision/ug/camera-calibration.html

[5] Zhang, Z. "A Flexible New Technique for Camera Calibration." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 22, No. 11, 2000, pp. 1330–1334.

[6] Computer and Machine Vision:Theory, Algorithms,Practicalities

Fourth Edition E. R. DAVIES Department of Physics Royal Holloway, University of London, Egham, Surrey, UK

[7] Accurate Non-Iterative O(n) Solution to the PnP Problem ∗

Francesc Moreno-Noguer Vincent Lepetit Pascal Fua

7 Appendix A

7.1 PC Technical Specifications

https://www.asus.com/us/Laptops/K55VD/specifications/

Processor:

Intel® Core™ i7 3670QM Processor

Operating System:

Windows 7 Home Premium

Chipset:

Intel® Chief River Chipset HM76

Memory:

DDR3 1600 MHz SDRAM, 2 x SO-DIMM socket for expansion up to 8 GB SDRAM

Display:

15.6" 16:9 HD (1366x768) LED Backlight

Graphic:

NVIDIA® GeForce® 610M with 2GB DDR3 VRAM

Storage:

2.5" SATA

1TB 5400RPM

750GB 5400/7200RPM

500GB 5400/7200RPM

320GB 5400RPM

Optical Drive:

Blu-Ray DVD Combo (Optional)

Super-Multi DVD :

Blue-ray Writer (Optional)

Card Reade:r

2 -in-1 card reader (SD/ MMC)

Camera:

0.3 Mega Pixel Fixed web camera

Networking:

Integrated 802.11 b/g/n

Built-in Bluetooth™ V3.0

10/100/1000 Base T

Interface:

1 x Microphone-in jack

1 x Headphone-out jack

1 x VGA port/Mini D-sub 15-pin for external monitor

2 x USB 3.0 port(s)

1 x USB 2.0 port(s)

1 x RJ45 LAN Jack for LAN insert

1 x HDMI

Audio:

Built-in Speakers And Microphone

Altec Lansing® Speakers

Battery:

6Cells 4700 mAh 50 Whrs

8 Curriculum Vitae

Omran Kaddah

Address: ilidrim.Mh oztekin.sk no.23.d5. Bahçeşehir / İstanbul

Mobile: **537 486 66 39**

E-mail: kadah.omran@stu.bahcesehir.edu.tr

Personal Informatio………………………..

Career Objective

Participate in software developer companies. Participate in Computer Vision research

Education

2011 - 2012 : Al-Saade High School - Damascus

2016 one semester at tu

20013 – 2017 : Bahcesehir University