COMPARISON OF 2D-3D POINT CORRESPONDENCES
2018

NANYANG TECHNOLOGICAL UNIVERSITY

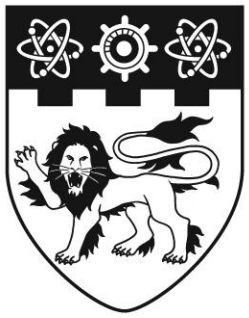
COMPARISON OF 2D-3D POINT CORRESPONDENCES

HANUMANTHA RAO SRINATH

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

2019

HANUMANTHA RAO SRINATH



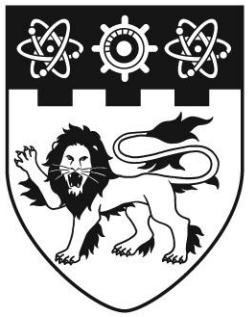
NANYANG TECHNOLOGICAL UNIVERSITY

COMPARISON OF 2D-3D POINT CORRESPONDENCES

HANUMANTHA RAO SRINATH

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

2019



NANYANG TECHNOLOGICAL UNIVERSITY

COMPARISON OF 2D-3D POINT CORRESPONDENCES

HANUMANTHA RAO SRINATH

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

A DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER CONTROL AND AUTOMATION

2019

Table of Contents:

Abstract

Pose estimation is one of the most basic steps when interfacing a camera with respect to the world. It has seen an increase in usage with the development of autonomy across various fields, mainly robotics. This thesis proposes to compare the various 2d-3d point correspondence methods for pose estimation available to us using three different conditions: distance from camera, resolution and pattern used.

There is no documentation available today that describes which pattern, distance and algorithm configuration gives us the best results possible. After getting various correspondences, a hand-eye calibration will be performed and then later used for testing which correspondence environment produces the most accurate results. Accuracy would be the metric used to calculate the results produced from the correspondence algorithms

Acknowledgements

I would like to express my sincere gratitude to Nanyang Technological University for providing me this opportunity to work on this thesis and enhance my academic understanding through the practical application during the course of this thesis.

I am hugely indebted to my project supervisors, Cheah Chien Chern, Associate Professor, School of EEE, Nanyang Technological University and Chen-I-Ming, Professor, School of MAE for allowing me to work on this problem statement through which I was able to update my knowledge in the field of Computer Vision and work with 3-D vision. This proved to be an opportunity for me to further my experience with ROS (Robot Operating System) and OpenCV (Library for Computer Vision) and also improve my skills in coding. I would also like to thank them for their reviews which helped me in completing my project successfully.

List of Figures

Chapter-1: Introduction

1.1 Background:

Pose estimation is one of the vital steps that needs to be performed for automation, especially in the field of robotics. It gives us two basic types of information:

a) Where is the object of interest located with respect to the camera/in the world coordinates?

b) What orientation does the object have?

Using the information above, we can interact with the object accordingly. These two types of information combine to give us DOF (Degrees of Freedom). A pose generally consists of 6 DOF – 3 to describe location (x, y, z) and 3 to describe orientation (roll, pitch, yaw). PnP refers to Perspective-n-Point problem where a set of 3D world points (points in the world) are mapped to the camera pixels so that the camera's position with respect to the object can be known. This is accomplished by using a planar/non-planar object that can be easily identified. In our thesis, only planar objects/patterns will be considered. For PnP to work, the camera in consideration should be calibrated so that the internal calibration parameters (fundamental matrix) is known. Using this information and the planar objects we can estimate the rotation and translation vectors required to transform the object's frame to the camera's frame allowing us to locate the camera in the world frame. This is shown in the picture below.

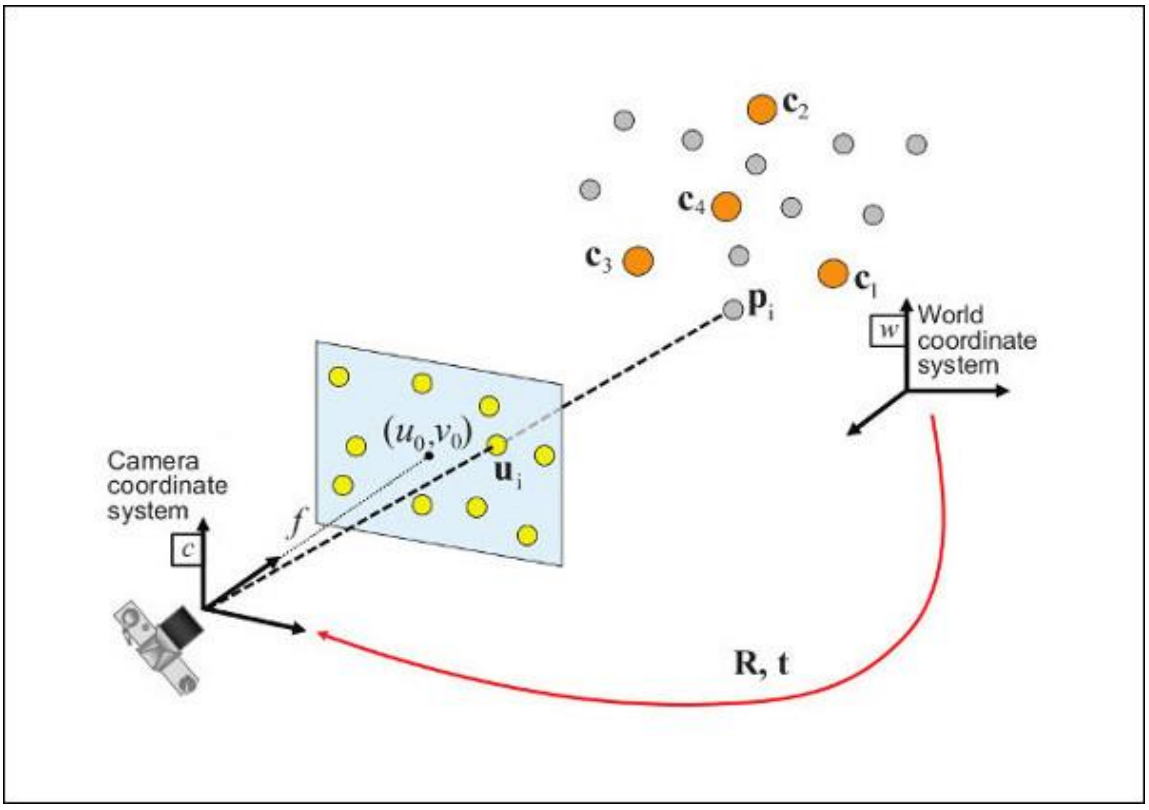


Figure 1: Finding rotation and translation vectors

In the picture above, R and t represent rotation and translation vectors respectively allowing the camera's position to be located with respect to the object. u_o and v_o represent the image points corresponding to the world points c1, c2, c3 and c4.

1.2 Motivation:

Although there have been advances in the algorithms used for PnP problems, there has been no documentation of how each algorithm performs under various conditions. This is important to know as it can help save time and resources when working with real-world problems. Companies need not waste their resources identifying which pose estimation method suits their needs and can implement the desired algorithm according to the

conditions stated. The implementation of these algorithms seems to go unnoticed when used in computer vision libraries such as OpenCV. The solvepnp method present in OpenCV uses the Iterative method as the default one. Users do not know how well the other algorithms perform on the same data they are using or they cannot judge therefore they seem to utilize the default method. The usage of patterns when solving the PnP problem is also a major question as there are various patterns available today: Checkerboards, Aruco markers, Ellipse markers and many more. There is again no documentation about the usage of markers and the corresponding accuracies derived from using them with various algorithms involved.

1.3 Objective and Scope:

The objective of this thesis is to identify the best working algorithms used in PnP solutions under various conditions. The varying conditions are:

a) Change in position (distance from camera to object)

b) Change in resolution

c) Change in markers

As every experiment is devised, we would also require control variables. The illumination, waypoints of the robot, robotic arm used, hand-eye calibration method used is kept constant to ensure these discrepancies don't affect the accuracy of the solutions.

After estimating the camera position for different waypoints, a hand-eye calibration is performed to test the accuracy of the calibration that directly informs us of the accuracy of the PnP algorithm/marker solutions.

1.5 Organization of the Dissertation:

Chapter 1: This chapter introduces and gives the basic information on pose estimation as well as the motivation and objectives to carry out the thesis.

Chapter 2: The literature review discusses the information/papers reviewed, which provided an insight on avoidance of pose estimation, hand-eye calibration, and the various PnP algorithms used.

Chapter 3: The experimental setup along with the hardware and software used are discussed in this section

Chapter 4: This section details the methodology of the experiment

Chapter 5: This section discusses the results and findings of the experiment

Chapter 5: The conclusion/summary and future work is discussed in this section

Chapter-2: Literature Review

The various information required to start working on pose estimation solutions are given below.

2.1 The Pinhole Camera Model and Internal Calibration

The pinhole camera was invented in the 20th century. It produces an inverted image on a film. We will be using the pinhole camera model to calibrate and get the intrinsic parameters of the camera. These parameters later are used to calculate the pose of the object.

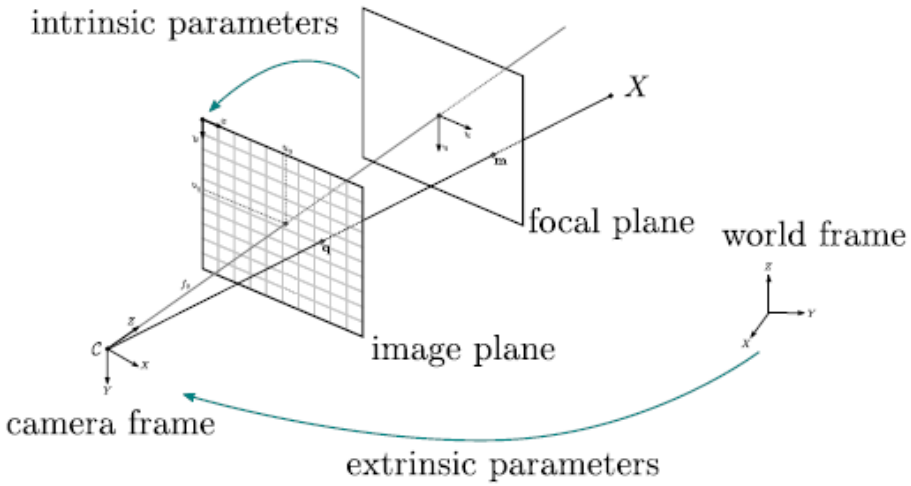


Figure 2: Intrinsic and Extrinsic parameters

As mentioned before, the extrinsic parameters are the rotation and translation vectors that are used to identify the pose of the given object. Internal calibration allows us to identify key special parameters. The intrinsic/internal calibration maps the focal plane(film) onto the camera hole.

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

The pinhole camera is characterized by the formula above [1]. The parameters f_x and f_y represent the focal length. The focal length is the distance between the film and the camera's pinhole in pixels. A perfect pinhole camera would not have the x and y components but would have just one focal length. The x_0 and y_o represent the offsets. The principal axis is a line that passes through the pinhole and intersects the film perpendicular to it. The x_0 and y_0 tell us about the offset of the centre of the film from the principal axis. Again, a perfect pinhole camera would not have any offsets. The process of calculating these values is called as internal calibration.

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (2)$$

In the equation above, we notice that the offsets are written as c_x and c_y. The image coordinates are considered as x and y, whereas the world coordinates are considered as X, Y and Z. w is mostly considered to be 1. The use of w is to ensure that the system is

homographic. Along with finding these characteristics it is important that we also reduce the distortion. The radial and tangential distortion is as given below

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (3)$$

Therefore, we have the following parameters to calculate

$$Distortion_{coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3) \quad (4)$$

These parameters are calculated by producing a set of simultaneous equations for each frame. A commonly used marker such as a checkerboard is fed as an image and the interspacing values are provided. Each picture provides a n equation for the above formulae allowing us to calculate the parameters by convergence.

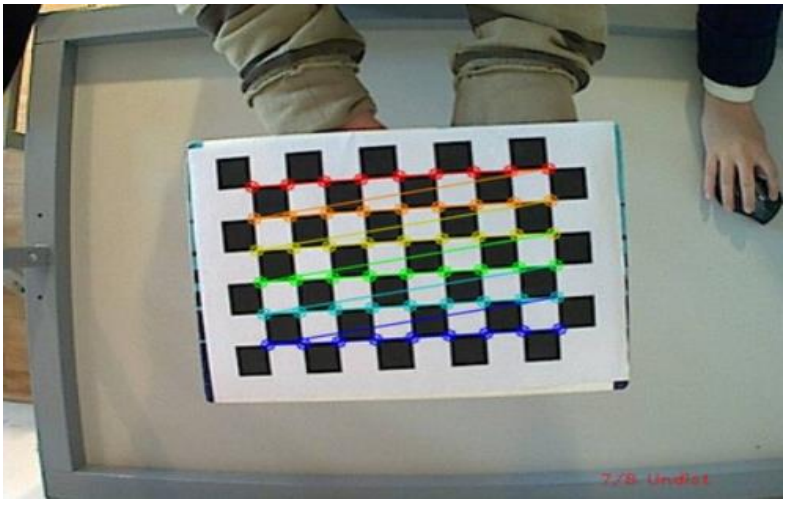


Figure 3: Marker for calculating intrinsic parameters

In the picture above, we can see that the points are identified on the image. These points are then mapped to the real-world coordinates and the parameters are calculated with ease by taking numerous pictures of the checkerboard at various angles to reduce distortion and increase accuracy of calculated values.

2.2 Hand-Eye Calibration

The hand-eye calibration allows us to locate the location of the camera in world coordinates. This helps us to easily identify the objects of interest located in the world corresponding to the ones in the image. Several pose estimation values are mapped along with the robot's location in the world to calculate the location of the camera frame in the world frame

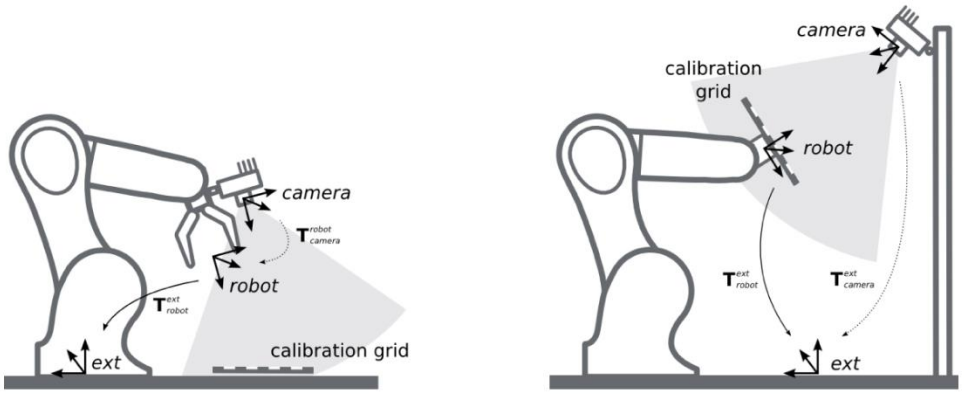


Figure 4: Configurations for calibration

Here, there are two configurations shown. The first configuration has the camera mounted on the robot whereas the second configuration has the grid/pattern mounted on the robot. The objective of the calibration is to identify where the real coordinates of the image points

are. The first configuration is called eye-in-hand and the second is called eye-of-hand. We will be utilizing the eye-of-hand configuration in this thesis.

The steps of a hand-eye-calibration are described below

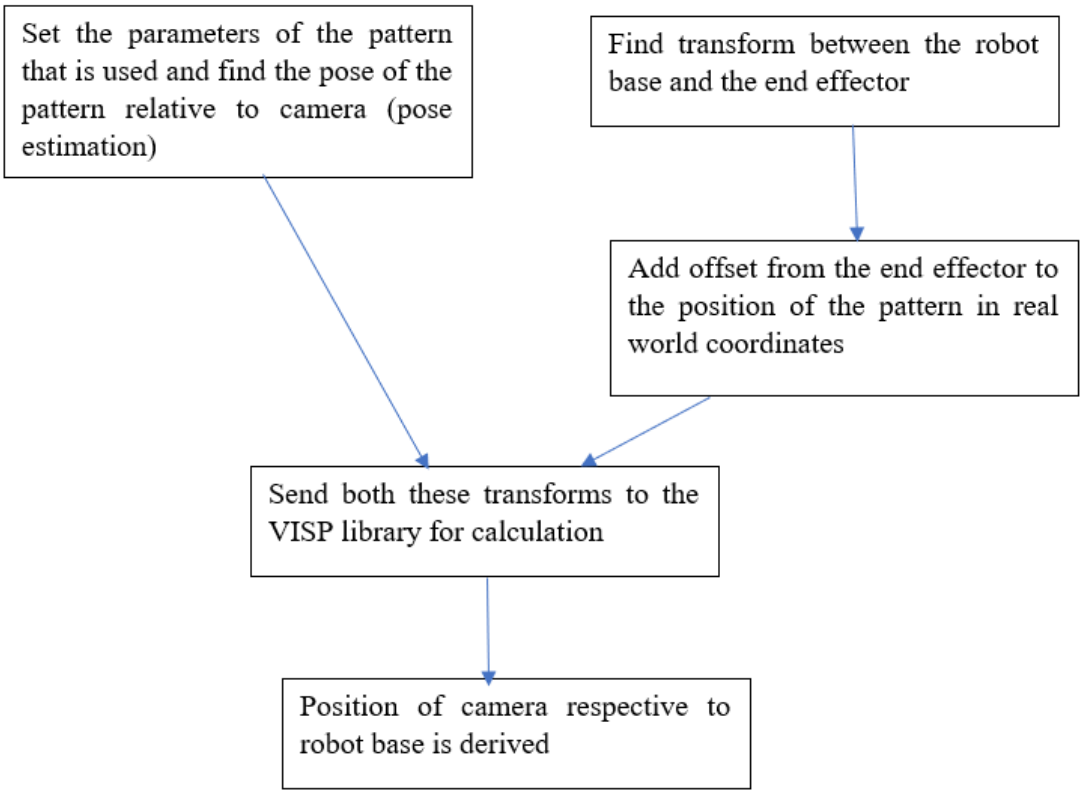
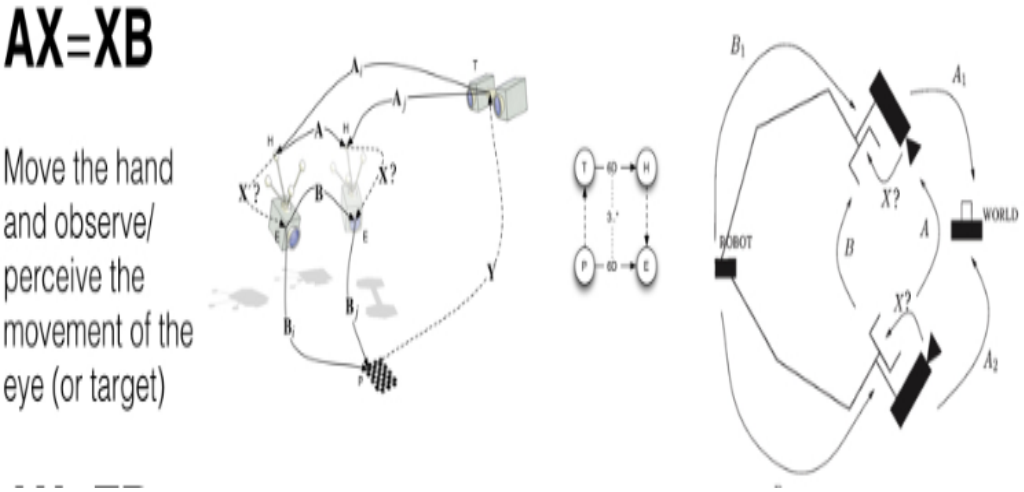


Figure 5: Workflow of hand-eye calibration

The pose estimation is first carried out to get the transformation of frames from the camera to the object/pattern. Later the transform from the robot base to the end effector is calculated, factoring in the offset from the end effector to the pattern being identified in the camera. Both these transformations are sent to the VISP library to output a

transformation that gives us the final position of the camera respective to the base of the robot.

Our solution for hand-eye is given as below



Where

A: Transform between camera and pattern

B: Transform between robot base and end offset

X: Transform between camera and robot base.

The transformations are calculated for a series of movements and positions of the pattern and sent to the VISP library [2]. The library computes the transformations and converges to provide us with one value of rotation and translation allowing us to identify where the camera is located in the real world with respect to the robot. The VISP library will not be discussed much in detail as it is out of the scope of this thesis.

2.3 Pose Estimation

To find a pose of the object(planar) we consider the object to be at no height(Z=0). This ensures that we find the position of the camera in space. After finding the pattern, we run the solvepnp function to get the rotation/translation vectors to get the transformation from camera to object/pattern. The process is described in a flowchart before.

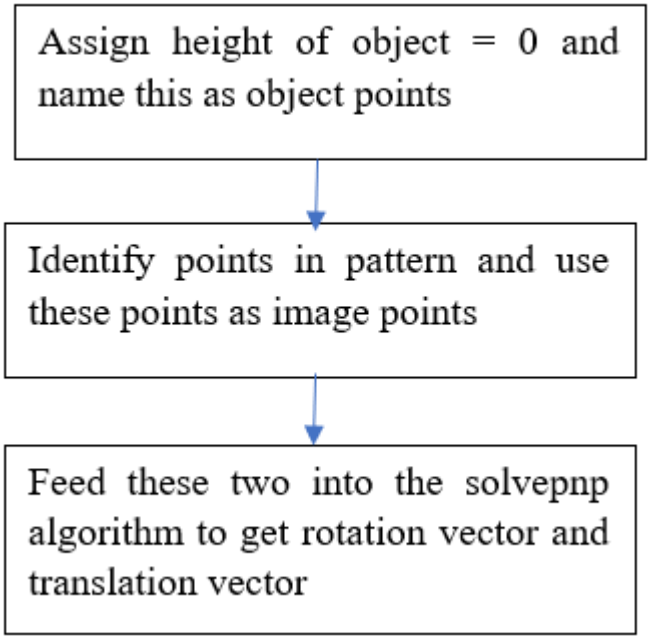
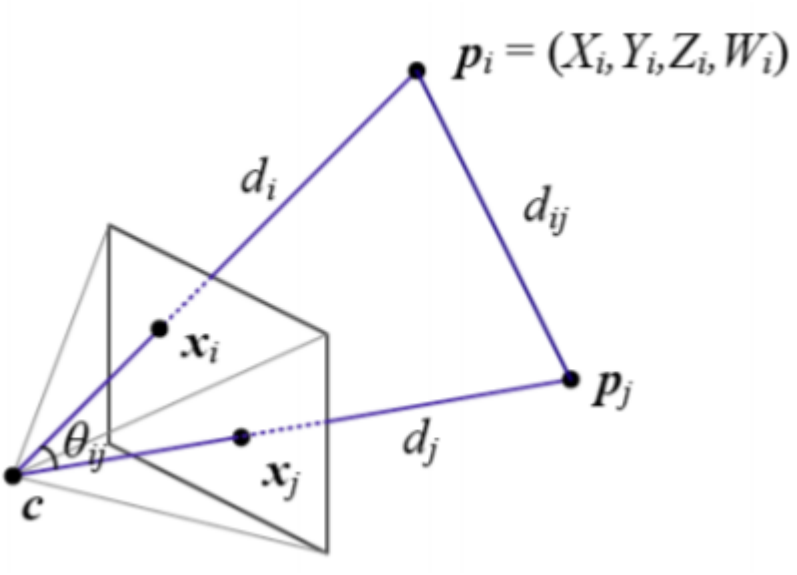


Figure 6: Flowchart to describe pose estimation algorithm structure

The solvepnp[3] method utilizes algorithms to produce the rotation and translation vectors. These algorithms will be discussed in further detail in the next section. We will be utilizing three different algorithms for the experiment. Levenberg-Marquardt Optimization, EPNP and P3P namely.

2.3.1 Levenberg Marquardt Optimization

This method[4] first utilizes a direct linear transform method and then moves on to use Levenberg Marquardt Optimization to ensure that the dof of R is reduced from 9 to 3.



There is a visual constraint here that the angle between two image points is the same as the angle between two points in the world coordinates. This angle helps us to find the vectors.

$$\begin{matrix} X \\ Y \\ Z \end{matrix} = R \begin{matrix} U \\ V \\ W \end{matrix} + t \qquad (5)$$

The equation described above is the final form to find extrinsic parameters. When expanded, the equation looks like the one below.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \tag{6}$$

If we find out a minimum number of correspondences between the image points and the world points we can solve for the r and t variables with a series of linear equations.

This is easily solvable but we notice an addition of s into the equation that appears from the intrinsic matrix. This complicates the equation and thus it cannot be solved directly.

$$s \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \tag{7}$$

We will thus employ a Direct Linear Transform[5] method here where we solve for the unknowns r and t. We can always use a DLT method when we see a linear equation is off by some scale.

After utilizing the DLT, we notice that our rotation vector has 9 degrees of freedom but we only require 3 (roll, pitch and yaw). One way of finding out the R and t vectors would be to minimize the reprojection error (sum of squared distances between 2d and 3d points).

We have to keep modifying R and t to ensure that the reprojection error reduces. This is easily achievable by using the Levenberg-Marquardt optimization technique that uses the Gradient descent method by finding the local minima.

2.3.2 EPnP (Efficient Perspective-n-Point) Algorithm

This method[6] utilizes a selection of 4 control points. The points can be chosen arbitrarily or taking the centroid of the reference points as one and the rest are chosen to form a basis that aligns with the principal directions. The equation below describes the control points

$$p_i^w = \sum_{j=1}^{4} \alpha_{ij} c_j^w, \quad with \sum_{j=1}^{4} \alpha_{ij} = 1$$

$$p_i^c = \sum_{j=1}^{4} \alpha_{ij} c_j^c, \quad with \sum_{j=1}^{4} \alpha_{ij} = 1 \tag{8}$$

P^w refers to control points in the world coordinates and P^c refers to control points in the camera coordinates. The objective is to compute the α which refer to barycentric coordinates. Our next step is to compute the coefficient of α using the formula below

$$\sum_{j=1}^{4} \alpha_{ij} f_v y_j^c + \alpha_{ij} (v_c - v_i) z_j^c = 0 \tag{9}$$

Once the coordinates are calculated, we use a Gaussian We have to utilize a Gaussian Newton iteration to reduce dimensionality and constraints to produce a stable result.

Before embarking on this, since our case is planar, we will be using the N=4 case where we have to derive M – whose kernel the solution must lie in. M is represented by

$$\mathbf{x} = \sum_{i=1}^{N} \beta_i \mathbf{v}_i \tag{10}$$

The Gaussian Newton optimization error should be minimized to give the formula present below to reduce error in β.

$$\text{Error}(\boldsymbol{\beta}) = \sum_{(i,j)\ \text{s.t.}\ i<j} \left(\|\mathbf{c}_i^c - \mathbf{c}_j^c\|^2 - \|\mathbf{c}_i^w - \mathbf{c}_j^w\|^2 \right), \tag{11}$$

After this step, we can get the best β and thus the best P^c and P^w and compute the Rotation and Translation vectors accordingly. We notice that in the planar case, the dimensionality of M is reduced to 2nx9 having 9D eigenvectors. Although the number of quadratic constraints drop from 6 to 3, we need to linearize again.

2.3.3 P3P

The P3P algorithm[7] only takes 3 correspondences between image and world points. Since it produces a lot of solutions, we use a 4th point to ensure ambiguity is removed in the solution and a stable result is produced.

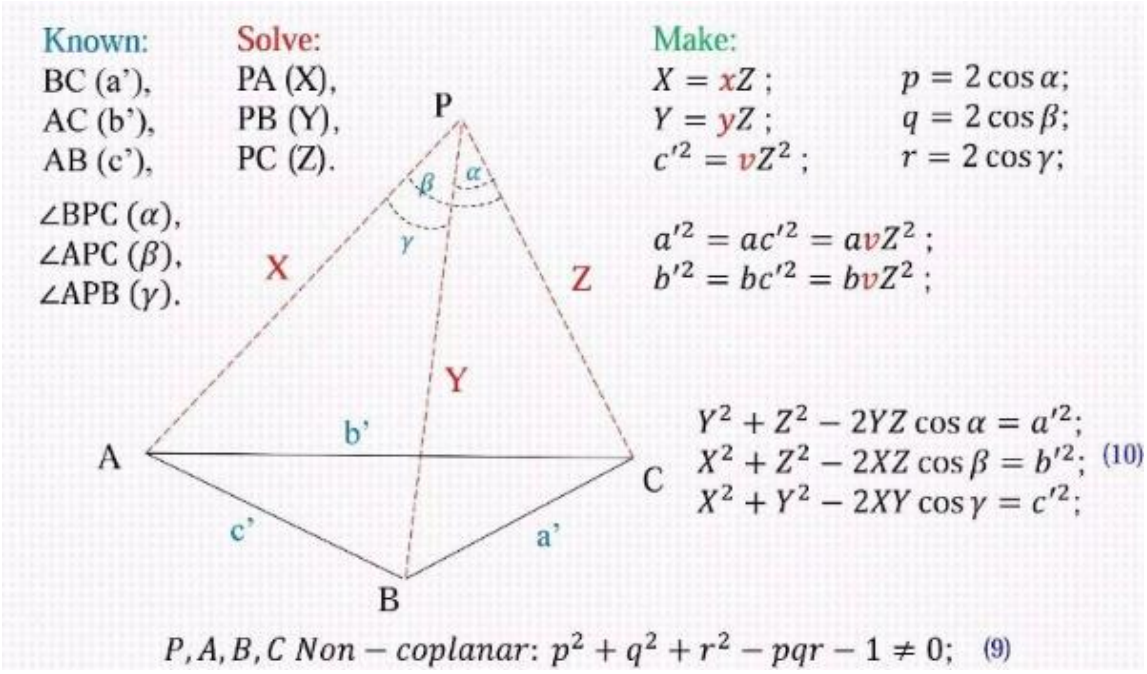


Figure 7: Equations for P3P

We must know that the points P,A, B and C are non-co-planar. Once this is done we use the law of cosines to get the formula below.

$$\begin{aligned} Y^2 + Z^2 - 2YZ\cos\alpha &= a'^2 \\ X^2 + Z^2 - 2XZ\cos\beta &= b'^2 \\ X^2 + Y^2 - 2XY\cos\gamma &= c'^2 \end{aligned} \tag{12}$$

The variables are replaced according to Figure 7.

$$\begin{aligned} y^2Z^2 + Z^2 - yZ^2p &= avZ^2; \\ x^2Z^2 + Z^2 - xZ^2q &= bvZ^2; \\ x^2Z^2 + y^2Z^2 - xyZ^2r &= vZ^2; \end{aligned} \quad (13)$$

Dividing by Z^2 we get

$$\begin{aligned} y^2 + 1 - yp - av &= 0; \\ x^2 + 1 - xq - bv &= 0; \\ x^2 + y^2 - xyr - v &= 0; \end{aligned} \quad (14)$$

Replacing v as $x^2 + y^2$

$$\begin{aligned} (1-a)y^2 - ax^2 + axyr - yp + 1 &= 0; \\ (1-b)x^2 - by^2 + bxyr - xq + 1 &= 0; \end{aligned} \quad (15)$$

$$g: b_0y - b_1 = 0; \quad f: a_0x^4 + a_1x^3 + a_2x^2 + a_3x + a_4 = 0; \quad (16)$$

(16) can have many solution so it is essential that we use a 4th point to determine the best answer.

$$Z = \frac{c'}{\sqrt{v}} \qquad \begin{matrix} X = xZ \\ Y = yZ \end{matrix} \tag{17}$$

Using 17 we can calculate R and t through matrix multiplication.

All the three methods produce outliers. To remove them and to not let these outliers affect the computation of rotation and translation vectors, we utilize a method called solvepnpransac that takes care of these.

Chapter 3: Experimental Setup

3.1 Hardware used

3.1.1 UR5 (Universal Robot 5) Robot Arm

The Universal Robot Arm 5 was used in experimentation for utilization in hand-eye calibration to ensure the accuracy of the algorithms and patterns used.

Figure 8: UR5 and teach pendant

The UR5 arm has the following specifications

- Payload: 5 kg,
- Weight: 20.6 kg,
- Reach: 850mm

The teach pendant that comes with the UR5 allows us to store certain values as waypoints. These values were later used as the control waypoints when calibrating the robot with the camera.

3.1.2 Zed Camera

The Zed stereovision camera was selected for its ease of use and ROS connectivity.

Figure 9:ZED Camera

The ZED camera has the following features

2.2K	15	4416x1242
1080p	30	3840x1080
720p	60	2560x720
WVGA	100	1344x376

Depth range – 0.5-20m

3.1.2.1 Stereovision

The zed camera uses stereovision to create 3d points in space. By comparing points from two lenses, a disparity map is produced. When both cameras view the same scene and are located at a distance “b” from each other same image points are produced at different

locations in the camera. This disparity is used to calculate the distance of the image point from the camera. This is better illustrated using a diagram below.

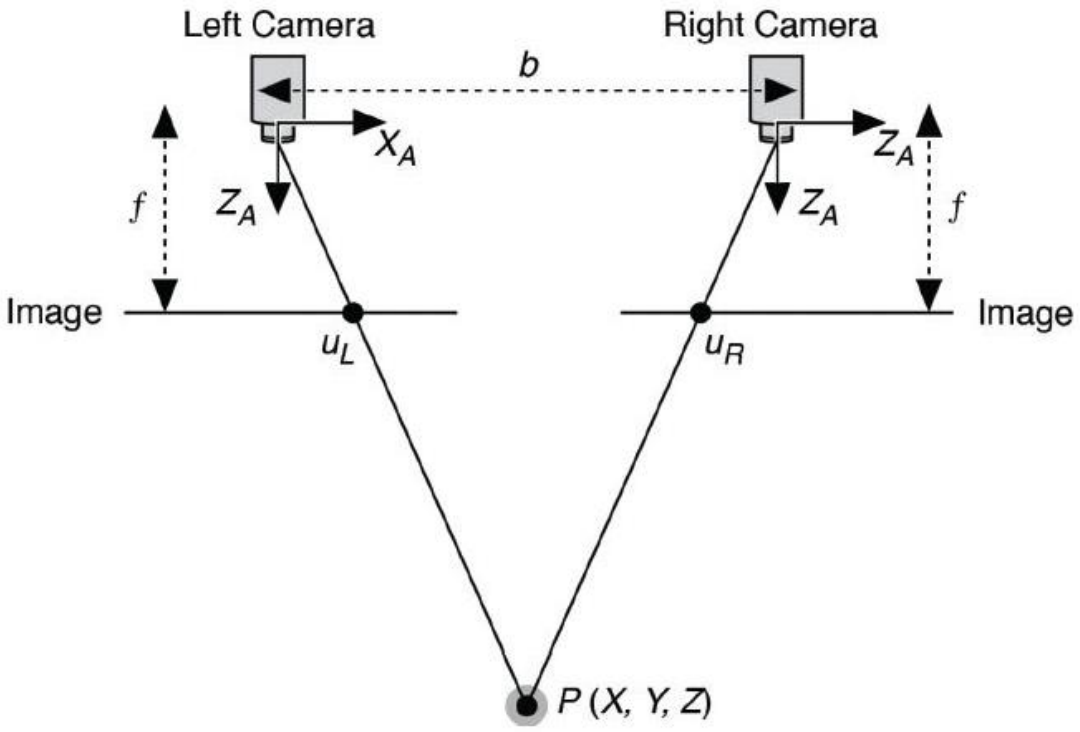


Figure 10: Stereovision

Stereovision is useful to create 3d points called pointclouds which will be discussed later.

3.2 Software and drivers used

3.2.1 ROS

The system to be calibrated requires a common ground for sending and receiving messages and data so that we don't need to create a separate communication channel and

perform all these tasks simultaneously. ROS (Robot Operating System) allows us to perform the functions listed above.

Ros introduces a system of nodes that allows executables(programs) to run simultaneously and be coupled (integrated). It acts as a middleware between the operating system and the devices/robots.

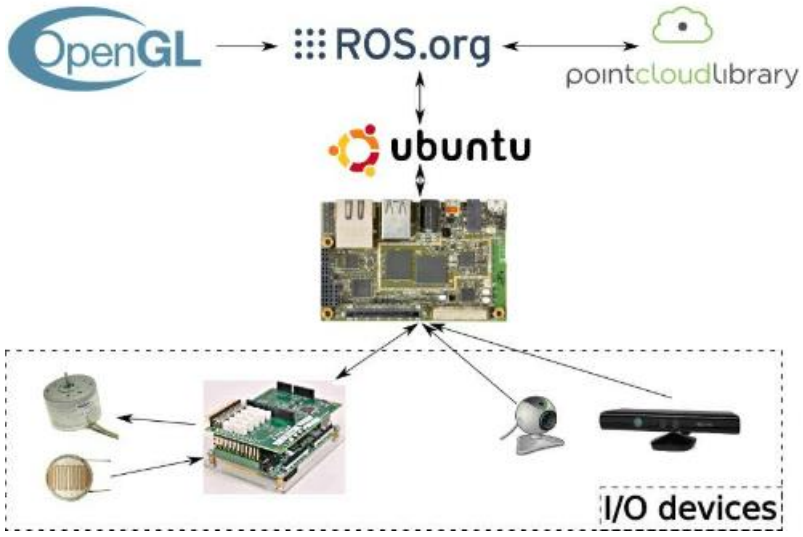


Figure 11: Example ROS flowchart

The picture above demonstrates an example of how a ROS process looks like. The software is able to work with a multitude of libraries making it easy for accessing and implementation while also connecting with a plethora of I/O devices.

3.2.2 OpenCV

OpenCV is a computer vision/image processing library that is used as a common infrastructure for computer vision applications and to accelerate use of machine learning.

OpenCV has a lot of functions built in and reduces the need to code every single mathematical equation that we need.

It can be used with both C++ and Python as well as interfacing with ROS helping systems to integrate with ease.

3.2.3 Pointclouds and PCL(Point cloud library)

Pointclouds are sets of data points that are outputs from 3d vision cameras. They are present in 3d space and have 4 major features (x,y,z and intensity).

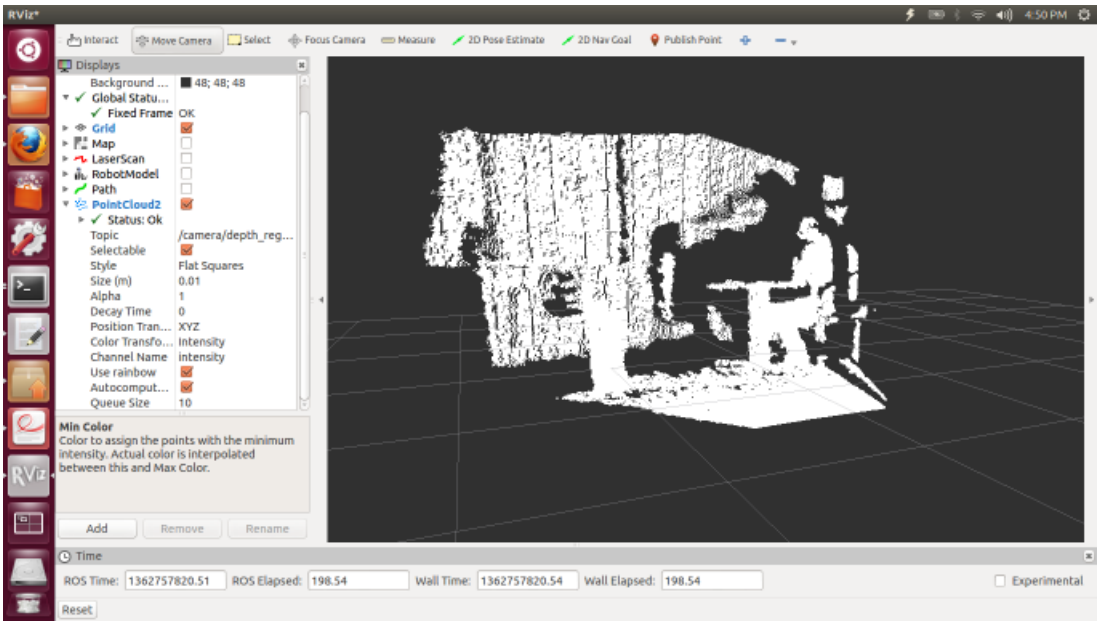


Figure 12: Pointcloud example

The pointcloud library helps us to interact and modify these 3d data points with ease. For our thesis we would require the pointcloud to verify if the algorithms are producing the pose with a good accuracy.

3.2.4 UR Driver

The UR driver can be interfaced with ROS. It helps to publish the robot model information and TF(Transformation trees). The TF trees can help us identify the transformation between the end effector and the base of the robot allowing us to provide it as an input to the hand-eye calibration model directly. An example of the robot model visualization and tf are given below.

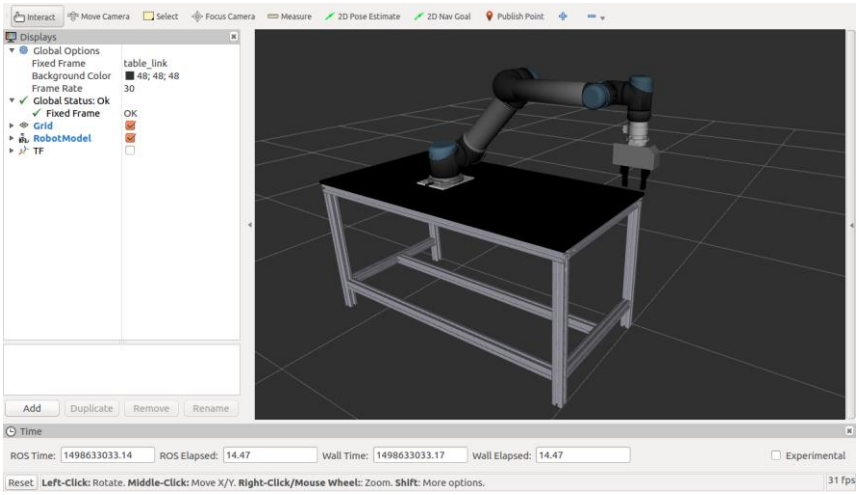


Figure 13:Robotmodel

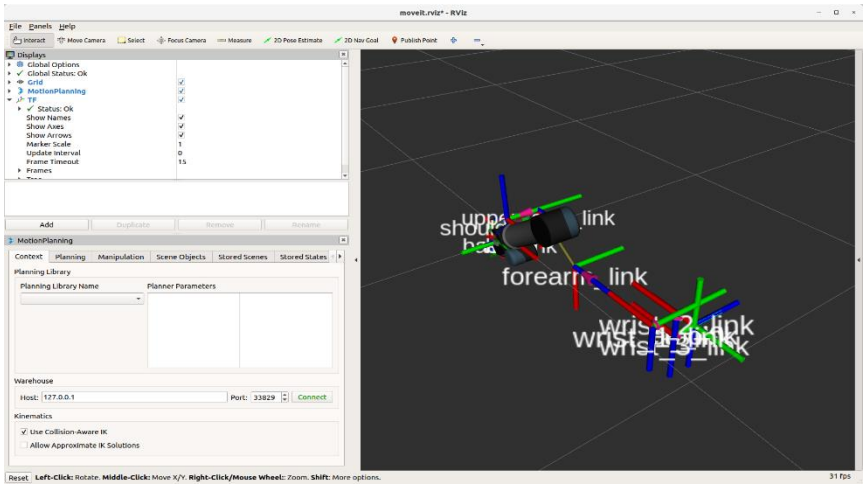


Figure 14:TF

3.3 Experimentation Setup

3.3.1 Setting waypoints

10 waypoints were chosen for calibration such that they spanned toward the extreme points of the camera's view. They are shown below

Figure 15: Waypoints

The waypoints are constant and do not change for any algorithm/pattern used. This is used to ensure that every calibration carried out has constant features and thus, can only be compared using the variables.

3.3.2 Position of Camera

Two fixed positions of the camera were taken into consideration.

a) 100cm from the base of the robot

b) 150cm from the base of the robot

The minimum distance for the camera to produce a good and stable point cloud was calculated to be 1m, thus these distances were taken accordingly.

The camera was placed on a tripod with an angle of 30 deg.

Figure 16:Camera Setup

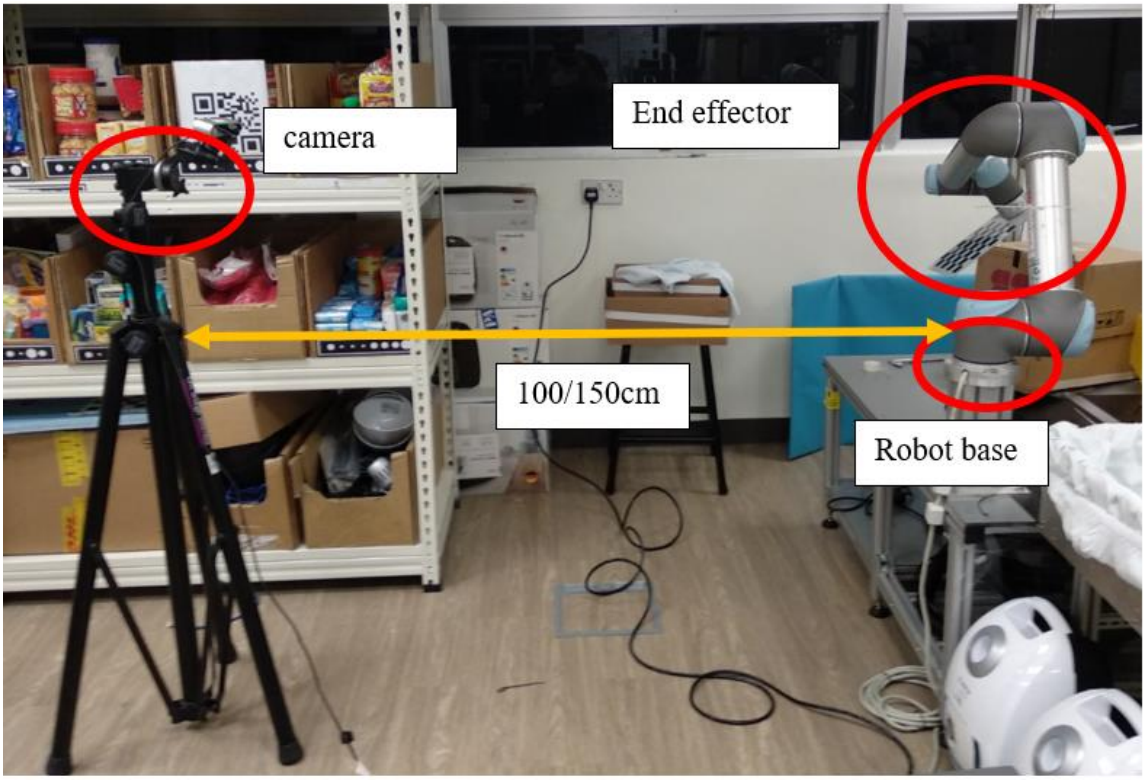


Figure 17: Experimental setup

3.3.3 Markers Used

There were 3 markers that were used.

a) Checkerboard

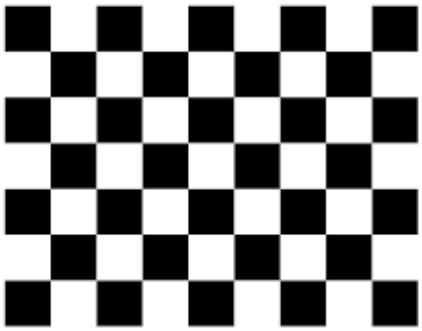


Figure 18:Checkerboard

Specifications:

i. 7x6 (width x height)
ii. size of square: 2.5cm

b) Aruco Marker

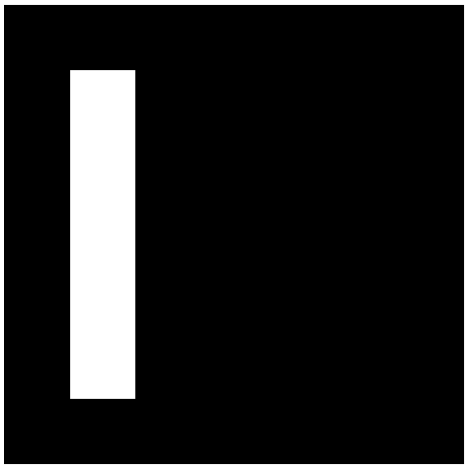


Figure 19:Aruco marker

Specifications:

i. size of each edge – 7cm

c) Ellipse marker

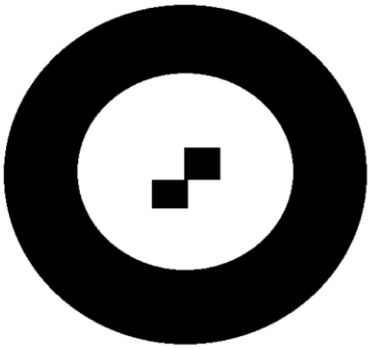


Figure 20:Ellipse marker

Specifications:

i. Outer circle diameter: 9.5cm
ii. Inner circle diameter: 6cm

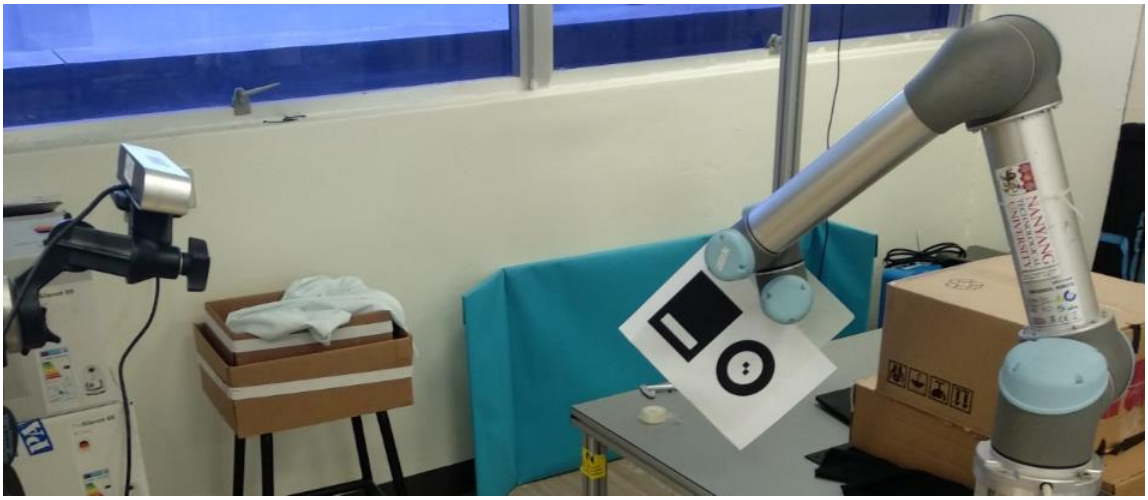


Figure 21: Experimental setup using other markers

Chapter 4: Methodology

After the experimental setup was complete, the setup was calibrated for different configurations and conditions.

4.1 Getting the transformation from robot base to end effector

The transformation from the base link of the robot and end effector is calculated using the Transform broadcaster[8] code present in ROS. After this, the offset from the end effector is added to the end effector to get the point at which the pose of the camera is calculated.

For the checkerboard, the pose is calculated at the right bottom square, therefore the offset is added so that the end effector is transferred to the exact point.

For the Aruco and ellipse markers, the pose is calculated at the centre of the markers, therefore the end effector is offset to the exact point.

These offsets are kept constant throughout and are assumed to be the true values so that the comparison between different configurations gives accurate results.

```
transformation.setOrigin( tf::Vector3(0.190,-0.075, 0) ); //for checkerboard
transformation.setOrigin( tf::Vector3(0.130,0.07, 0) ); //for aruco
transformation.setOrigin( tf::Vector3(0.130,-0.06, 0) ); // for ellipse
```

Figure 22: Code snippet for offsetting pose points

4.2 Parameters Extraction using 2D camera

For checkerboard: All the corners of the checkerboard are detected and stored as the image points. These are fed to the PnP algorithm along with a set of object points keeping the Z as 0 and the width of the points equal to the size of each square.

For Aruco: Aruco markers have a dictionary. Our Aruco used is present in the original Aruco dictionary, the corners of the detected Aruco are taken as the image points and the object points are initialized as the 4 corners with width as specified by the user.

For Ellipse: The two concentric circles provide us with the same centre and the two squares present in the middle provide the orientation. The circles are taken as the image point and the object points are taken with circumference range given.

4.3 Pose Estimation using solvepnp function

The pose for each type of marker is estimated using the image points, object points and the type of algorithm used. Along with these input parameters we should also provide the intrinsic parameters that we obtained from normal internal calibration.

```
cv2.solvePnP(objectPoints, imagePoints, cameraMatrix, distCoeffs[, rvec[, tvec[, useExtrinsicGuess[, flags]]]])
```

Figure 23:Python function for solvepnp

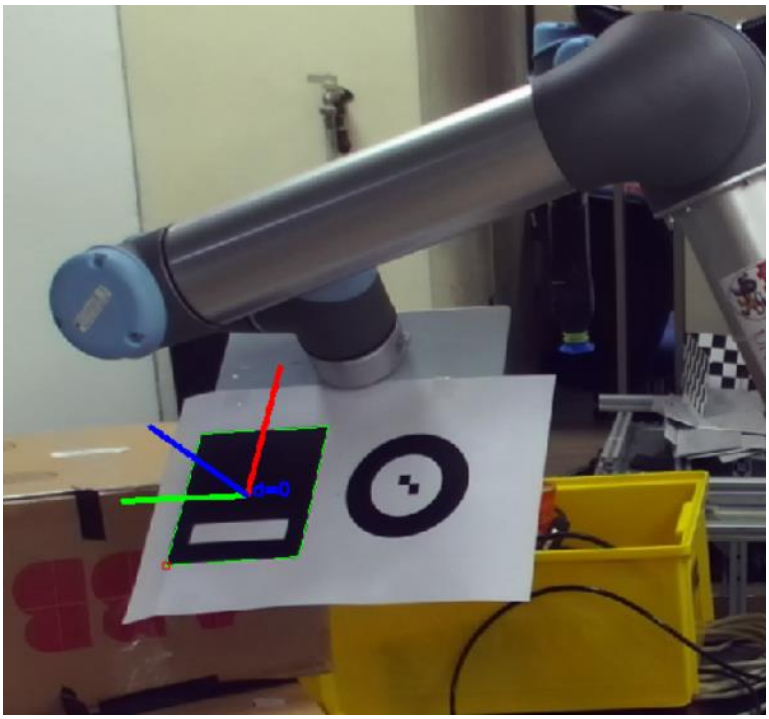


Figure 24:Example Pose estimation for Aruco

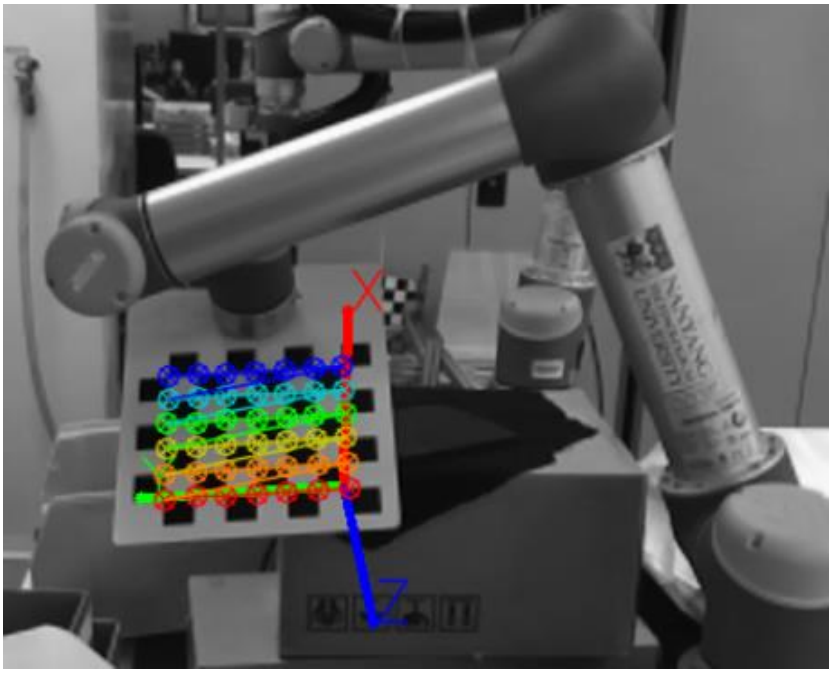


Figure 25:Example pose estimation for checkerboard

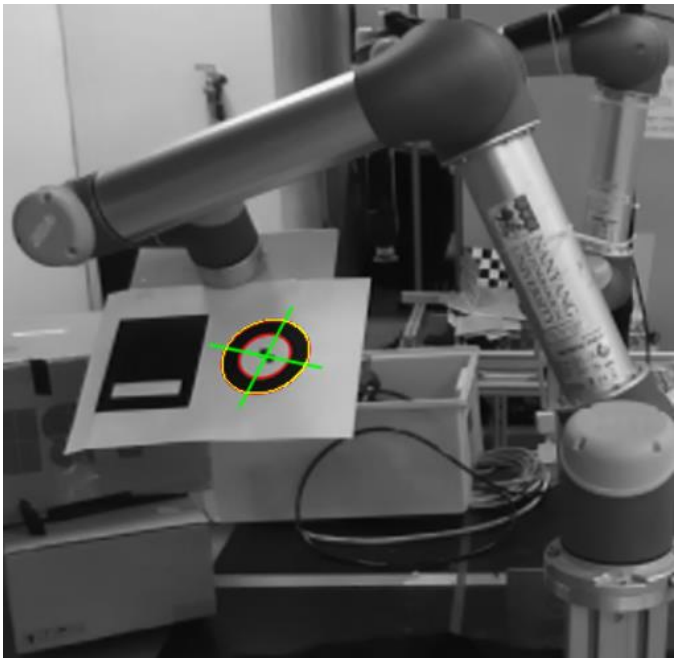


Figure 26: Example pose estimation for ellipse

.

The pose obtained gives us the transformation from the camera to the object and can be mapped in 3d space as a point cloud. The output of a solvepnp function are the rotation and translation vectors which we can use to visualize the pose in 3D.

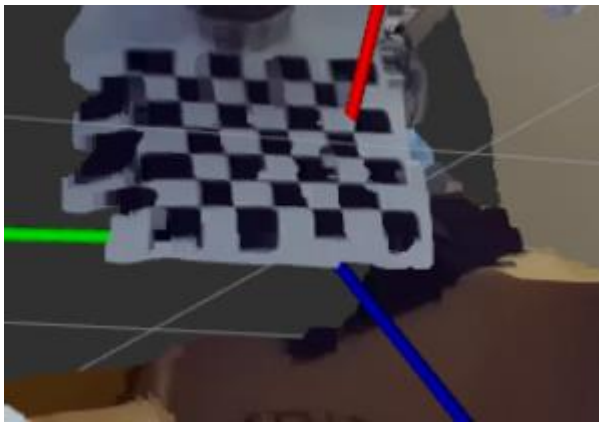


Figure 27: Pose in 3d point cloud

4.4 Errors Encountered with Change in Resolution/Algorithm/Position

Resolution error:

When testing out different resolutions, the VGA (lowest resolution of the camera) was unable to detect checkerboards even at the optimal distance from the robot

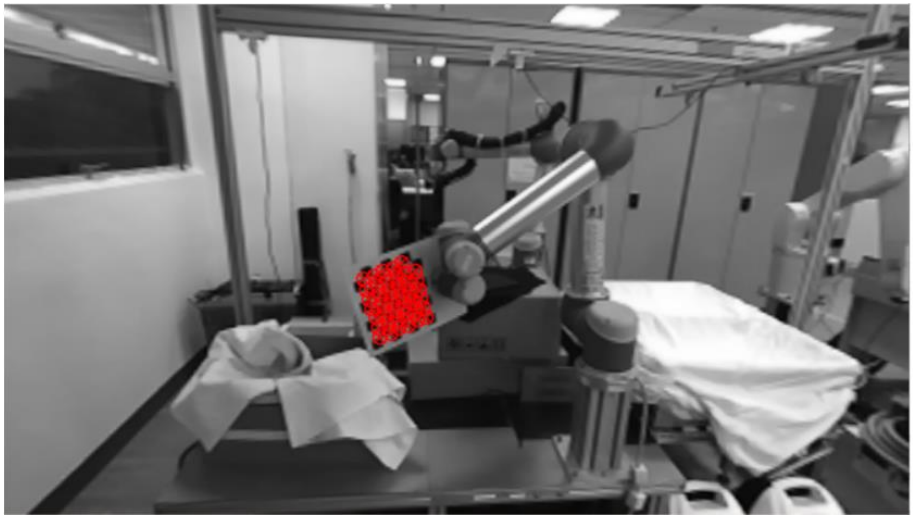


Figure 28:low resolution error

Therefore, this resolution was not taken into consideration while testing out the different algorithms and markers.

Algorithm Implementation error in OpenCV:

When implementing the EPnP and P3P algorithms with OpenCV on checkerboard markers, it was identified that the pose produced in both the 2d and 3d data spaces were wrong, i.e the x and z vectors of the orientation were switched. In the picture below x,y and z refer to red, blue and green. We notice that the red(x) is pointing away from the board which actually signifies the “z” direction.

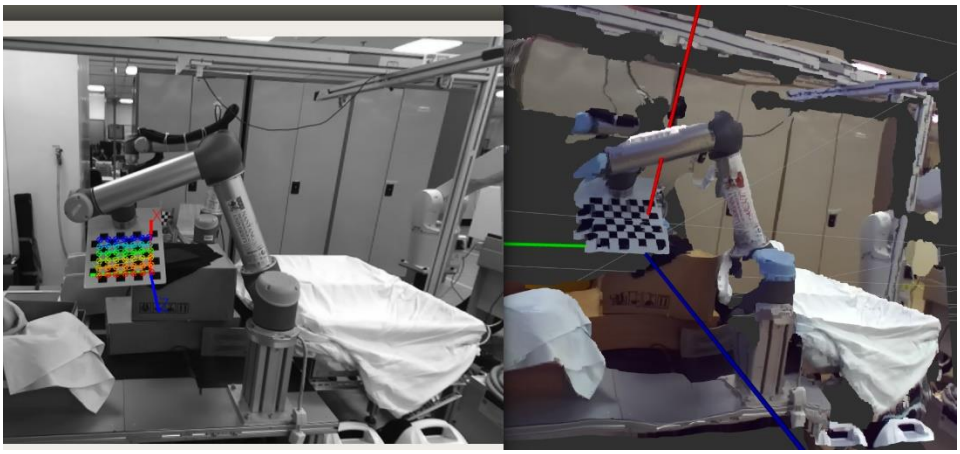


Figure 29:Error in Algorithm

To rectify this, the quaternions were inverted to ensure that the orientation was correct.

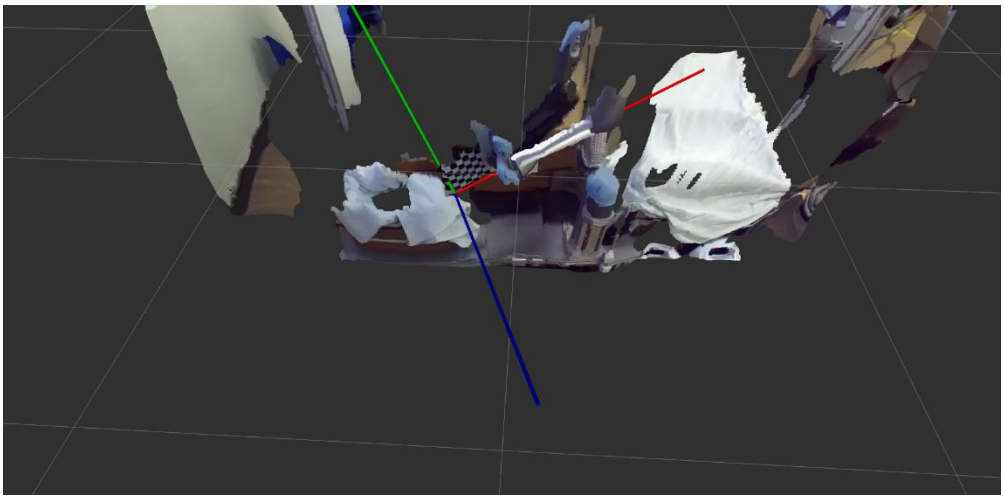


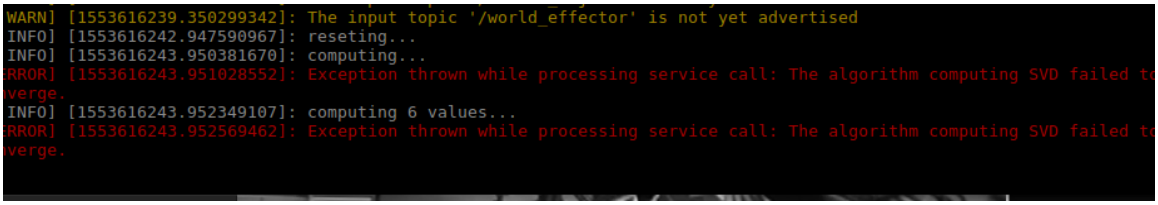
Figure 30:Error rectified

Ellipse errors:

It was noticed that the solvepnp function did not converge for P3P and EPNP when used with the Ellipse marker. Upon further research, the OpenCV implementation was broken for this specific case and thus could not be researched on further.

Distance Errors:

When placed at position 2, the checkerboards were failed to be detected and thus, data could not be collected further. If data was collected for some waypoints, the hand-eye library could not converge to give a stable result for the position of the camera.



4.5 Hand-eye-calibration

After the pose is identified for all the waypoints, the two transforms are fed into the VISP[9] library to get an output consisting of location and orientation of the camera in the world plane where the robot is situated.

```
Translation: [-0.423, 0.276, 0.076]
Rotation: in Quaternion [-0.383, 0.817, 0.412, 0.126]
          in RPY (radian) [2.400, 0.548, -2.483]
          in RPY (degree) [137.515, 31.411, -142.255]
```

Figure 31: Hand-eye calibration output

4.6 Measuring Accuracy

After hand-eye calibration, the camera frame is mapped to the world frame and the transformations are plotted on the 3D data space. The 3-d distance between the two points are calculated to find out the accuracy.

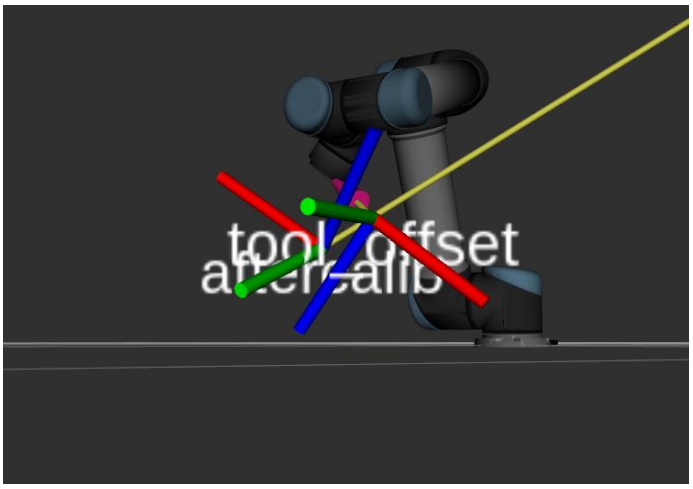


Figure 32:Accuracy measurement

4.7 Workflow for Experimentation

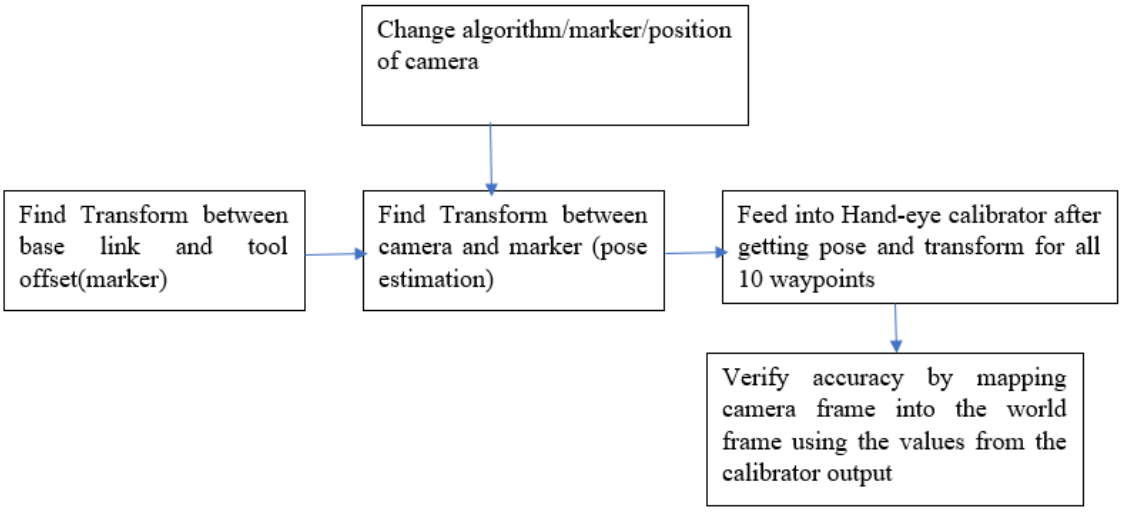


Figure 33:Workflow for Experimentation

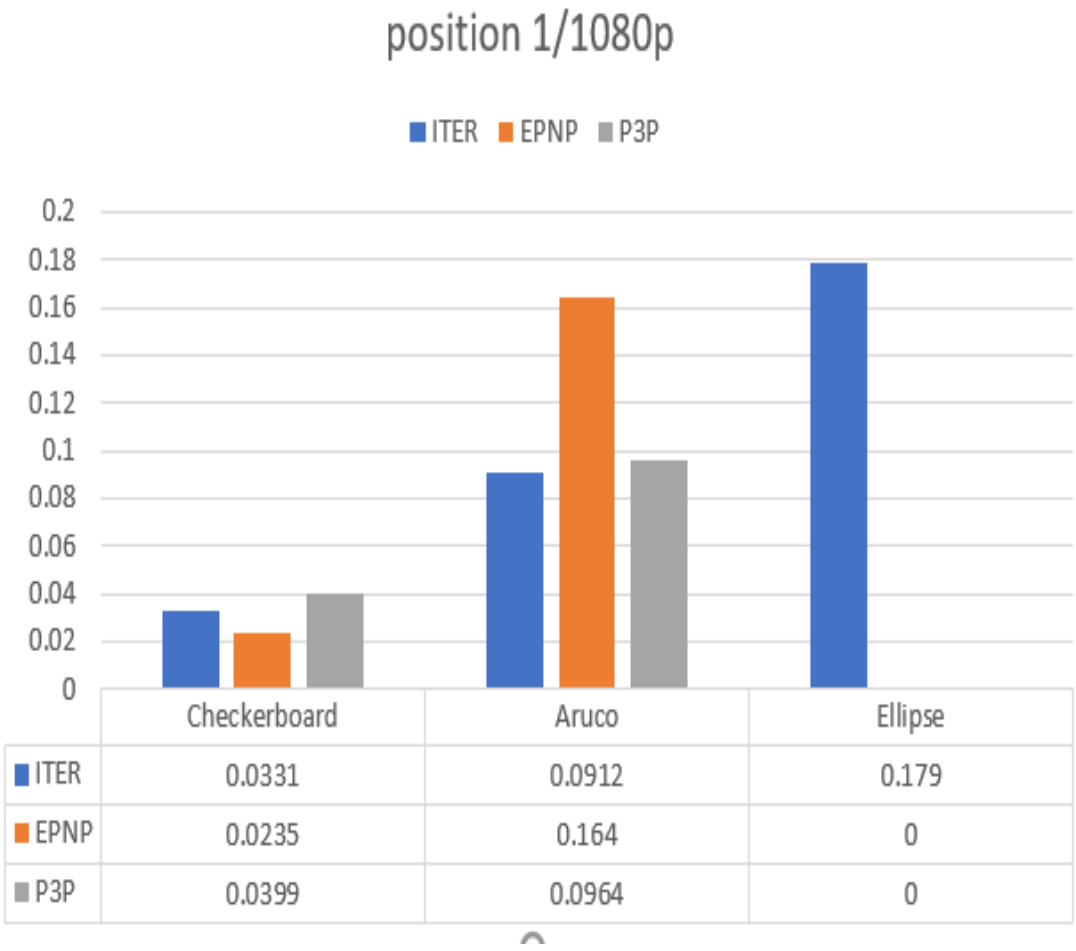
Chapter 5: Results and Discussion

All graphs represented demonstrate the distance between the real world coordinates and the estimated coordinates from calibration(lower is better)

	Checkerboard	Aruco	Ellipse
ITER	0.0326	0.0954	0.179
EPNP	0.0323	0.0818	0
P3P	0.0289	0.0868	0

Figure 34:Position1/720p values

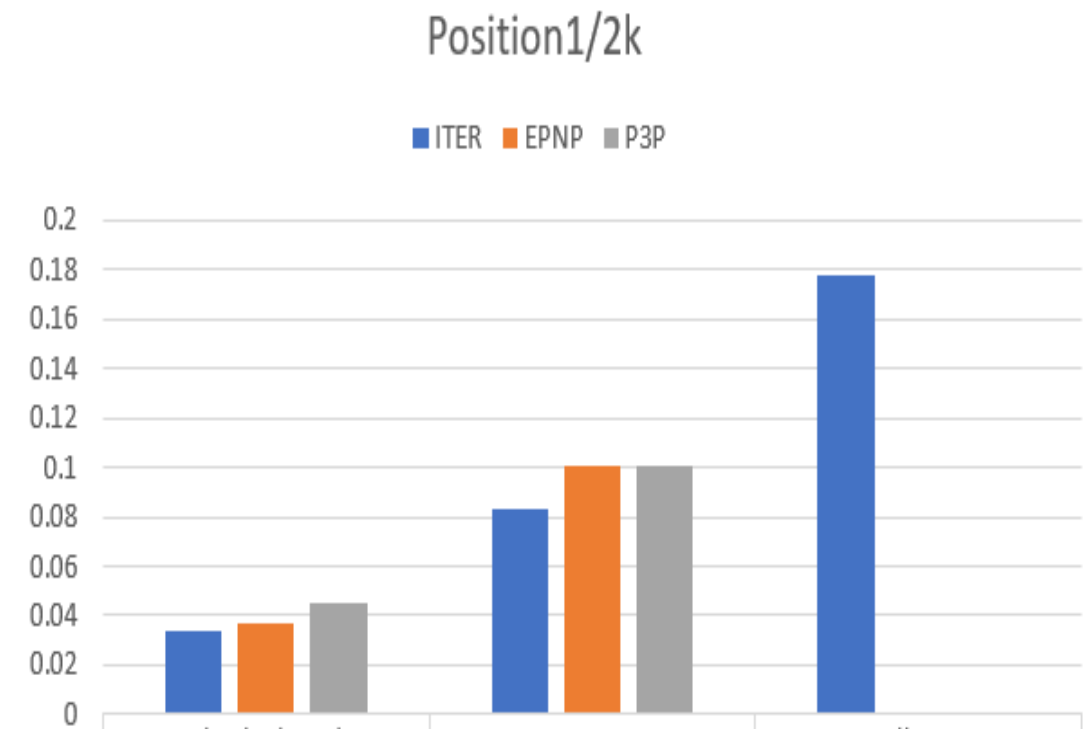
We notice that the accuracy is the highest while using the Checkerboard for all algorithms used. While comparing the algorithms we see that P3P performs the best while using a checkerboard and EPnP while using an Aruco board, but the accuracy is far greater while using a checkerboard.



	Checkerboard	Aruco	Ellipse
ITER	0.0331	0.0912	0.179
EPNP	0.0235	0.164	0
P3P	0.0399	0.0964	0

Figure 35:Position1/1080p values

We can notice that again, the accuracy is far greater while using a checkerboard as compared to other markers but when we compare the accuracy of the algorithms but this time, EPnP gives us the best accuracy while using a checkerboard whereas Iterative gives us the best while using the Aruco.



	Checkerboard	Aruco	Ellipse
ITER	0.0333	0.0832	0.178
EPNP	0.0364	0.101	0
P3P	0.0449	0.101	0

Figure 36:Position 1/2k

While using the maximum resolution, we notice that the Checkerboard provides the best accuracy again but this time, the iterative algorithm produces the best accuracy while using the checkerboard. Again, we see that the iterative produces the best accuracy while using the Aruco board at the highest resolution.

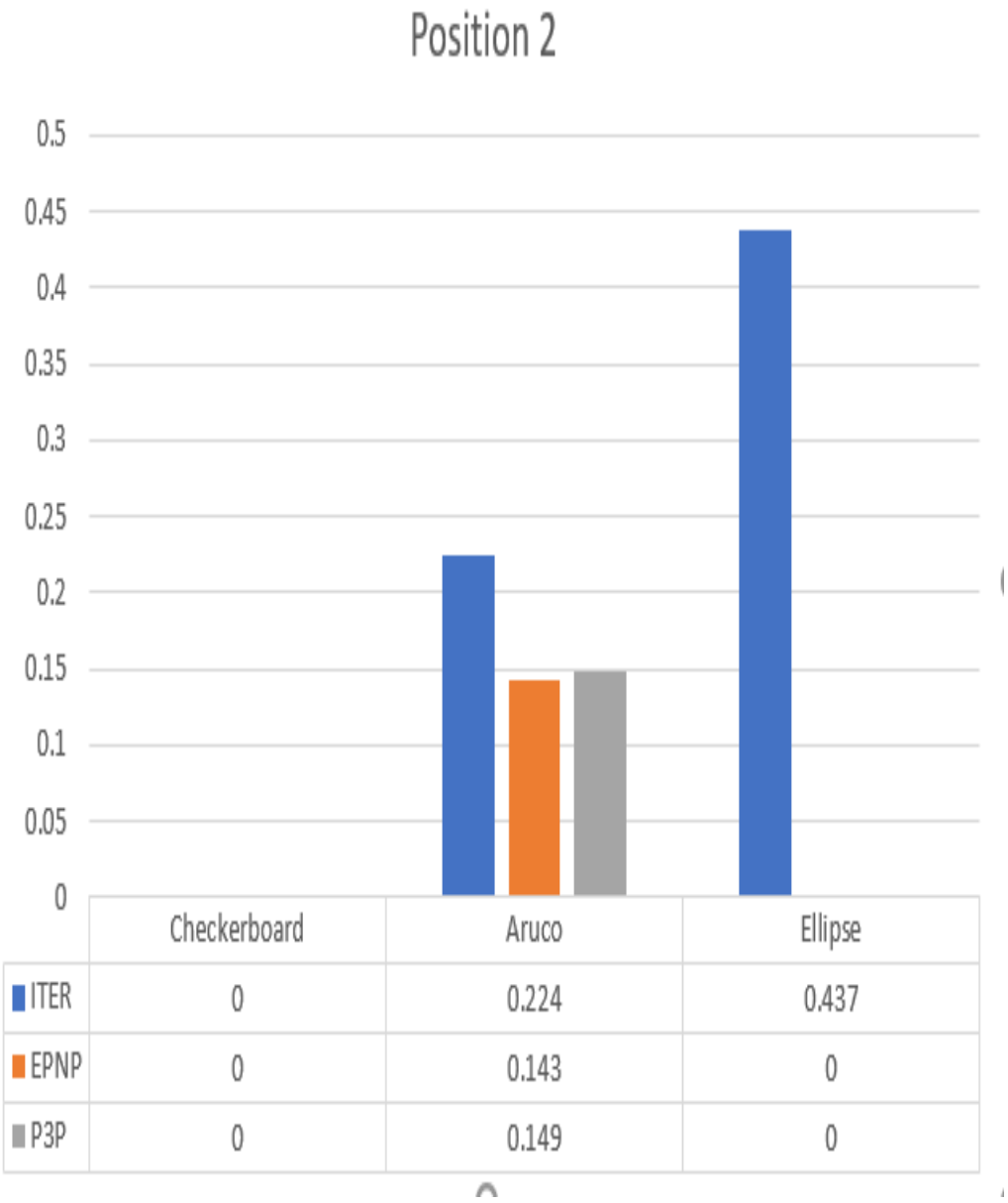


Figure 37:Position2

At position 2(150cm) of the camera, we notice that the checkerboard cannot be detected Also Aruco produces the best accuracy at this range. The EPnP algorithm fares the best at this position of the camera

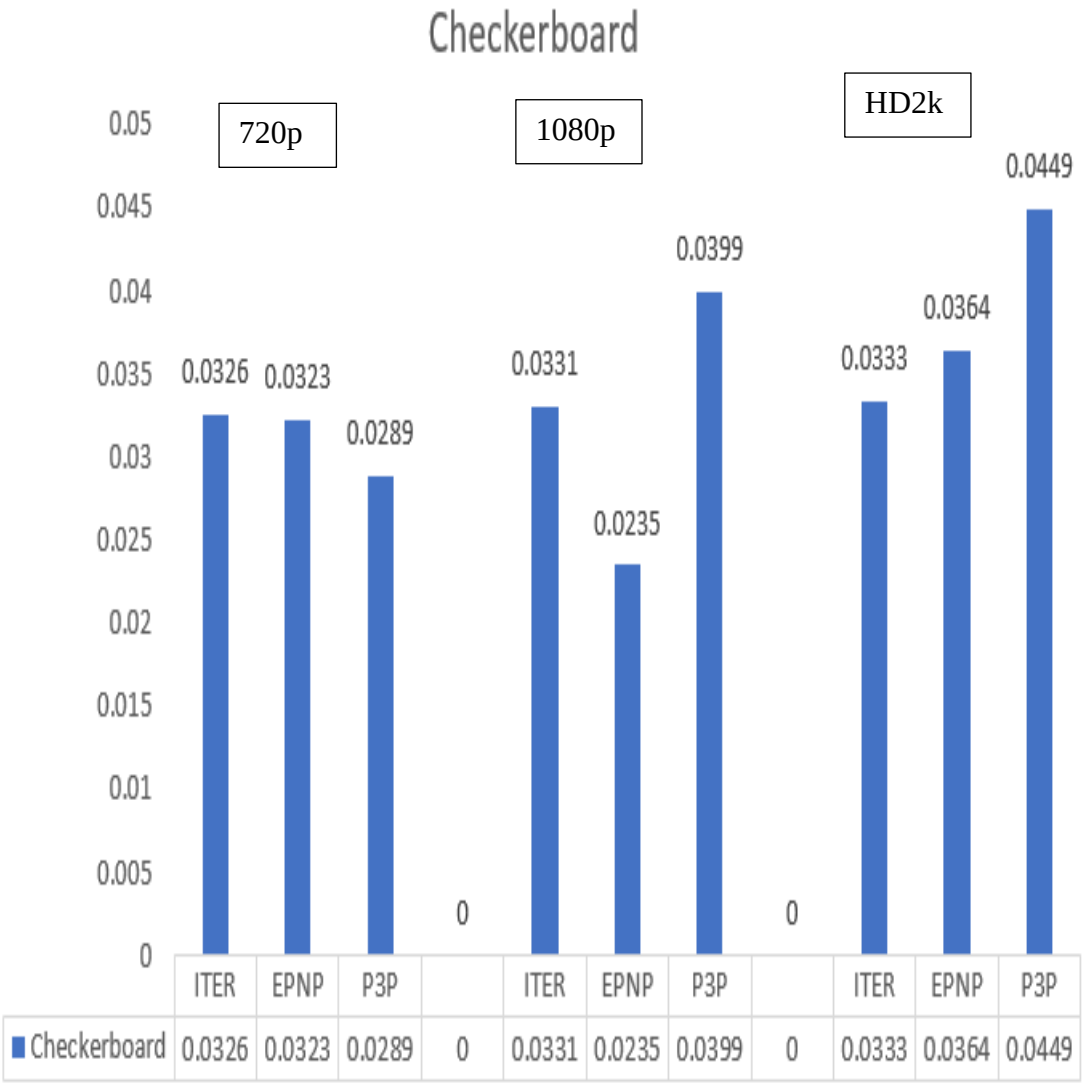


Figure 38:Compare Checkerboard

We notice that the highest accuracy is achieved when the resolution is at 1080p using the EPnP algorithm. For the Iterative and P3P algorithm, the best accuracy is achieved at 720p. We notice a decrease in accuracy as the resolution increases.

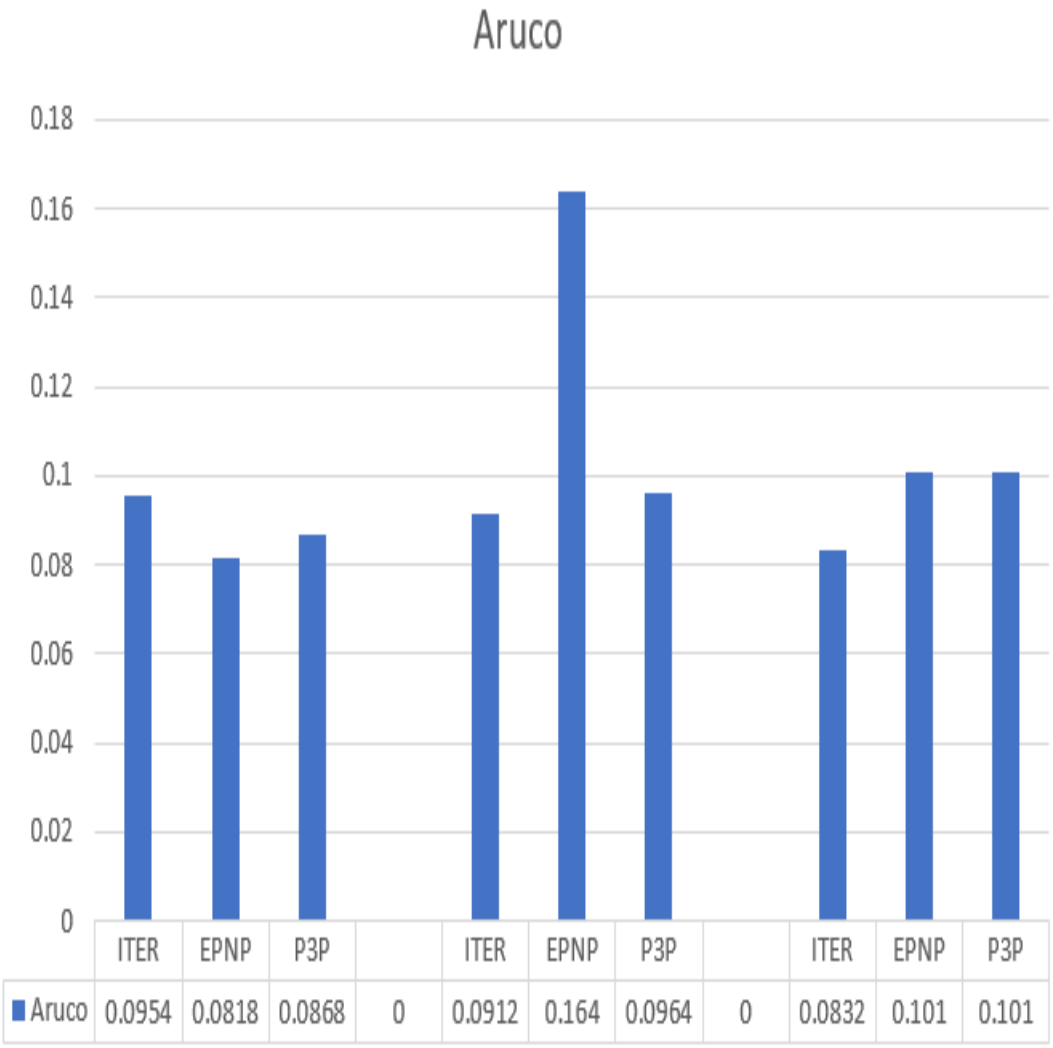


Figure 39:Compare Aruco

When using the Aruco marker, we can notice that the best accuracy is achieved at 720p using the EPnP algorithm. If the 1080p or HD2k resolution is used, the Iterative algorithm produces the best accuracy. We also see a similar trend where accuracy reduces as the resolution increases when using the EPnP and P3P algorithm

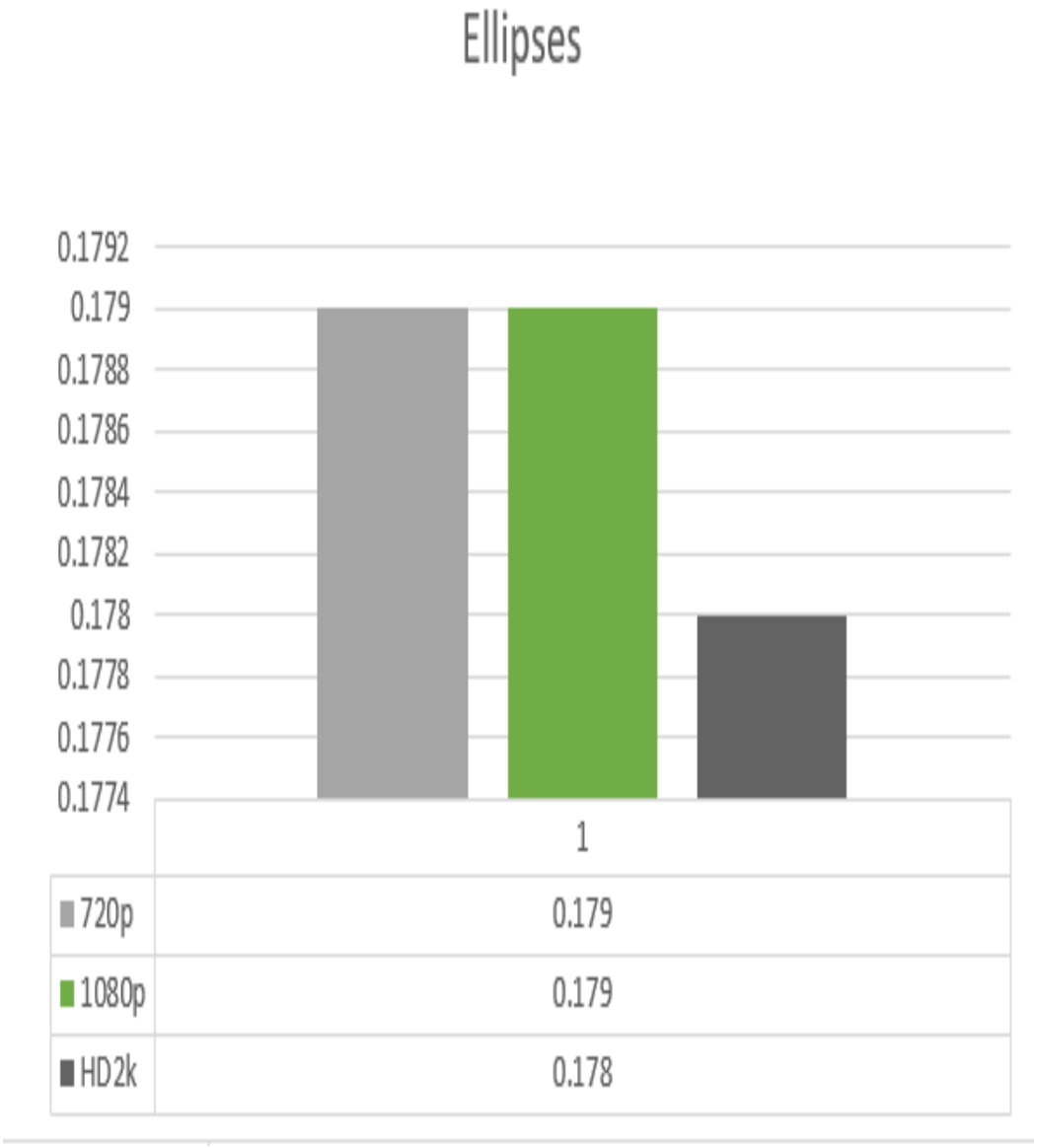


Figure 40:Ellipses Compare

When using the Ellipses marker we notice that all the three resolutions produce similar accuracies but the accuracies are very low. (error of 17cm)

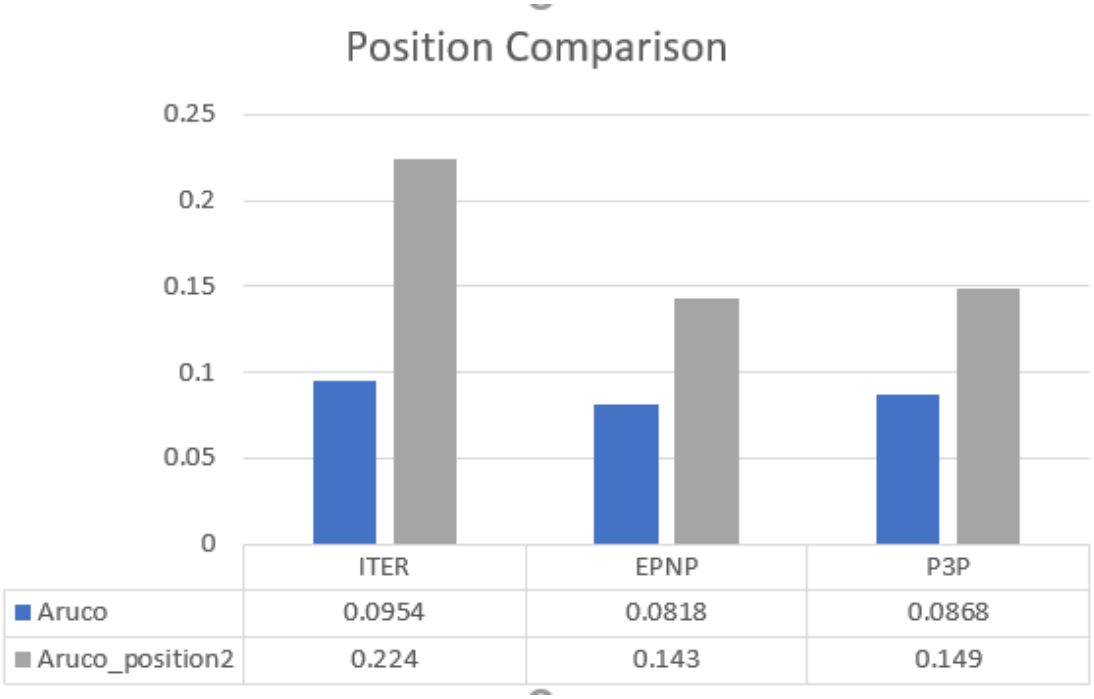


Figure 41:Position compare

When comparing between different positions, we notice that only the Aruco marker is able to be identified at both these positions. When comparing between the positions we notice that as we go further away from the robot, the accuracy decreases. EPnP produces the best results as it is tested under 720p resolution.

Chapter 6- Conclusion and Future work

6.1 Summary

In this dissertation, different Point correspondence methods were compared by testing them under various conditions and with different markers. It was noticed that for certain conditions, certain algorithms and markers worked well. The checkerboard marker works the best when the distance is comparable(50-120cm) from the robot. If the distance increases, we notice that the camera is not able to identify the checkerboard squares. The ellipses produce the least accuracy while calibrating. This might be due to the fact that the orientation of the ellipse was misaligned to the board. Therefore, it is essential that the orientation of the Ellipse marker always aligns with the board without any deviation. This might produce better results if tested in the future. Also the Ellipses were not able to work with other algorithms such as P3P and EPnP and therefore need to be fixed in the OpenCV library. The Aruco marker was able to be detected at both the locations and produced considerable accuracy at both. For easy and fast implementation, Aruco marker can be prescribed while using the respective algorithm at the desired resolution. The resolution played a role in accuracy. With increase in resolution we could notice a decrease in accuracy of the calibration. This might be due to the fact that the algorithm had to search through more points when detecting the markers. The position of the camera has an inverse relation with the accuracy of calibration. As we moved the camera away, we could see a decrease in accuracy of both Aruco and Ellipse marker setups. There was no clear winner while using the algorithms as each algorithm produced good results under different conditions. It was henceforth proved that we need not only rely on the Iterative algorithm

that the solvepnp function uses by default and we can use the information given above to choose our algorithm and marker according to the needs of our objective.

6.2: Future Work

Once the P3P and EPnP functions are fixed in the OpenCV library we can test these algorithms with the Ellipse markers thus allowing us to have more choices while selecting an algorithm. More conditions can be taken into account while trying to setup experiments such as illumination, noise in images and eye-in hand configurations while performing calibration. This thesis has considered only algorithms that are common and widely used. Other algorithms such as UPnP can be also tested and compared for a greater choice. Along with testing the accuracy, the speed of calibration and detection can also be tested. Other markers can also be used such as QR codes. This will help develop a dataset that the community can use while using point correspondence algorithms.

Bibliography

1. V. Popescu, B. Benes, P. Rosen, J. Cui and L. Wang, "A Flexible Pinhole Camera Model for Coherent Nonuniform Sampling" in IEEE Computer Graphics and Applications, vol. 34, no. 04, pp. 30-41, 2014.

2. E. Marchand, "VISP: a software environment for eye-in-hand visual servoing," Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), Detroit, MI, USA, 1999, pp. 3224-3229 vol.4.
doi: 10.1109/ROBOT.1999.774089

3. "Camera Calibration and 3D Reconstruction¶." *Camera Calibration and 3D ReconstructionOpenCV2.4.13.7Documentation*,docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=solvepnp.

4. T. Shao-xiong, L. Shan and L. Zong-ming, "Levenberg-Marquardt algorithm based nonlinear optimization of camera calibration for relative measurement," *2015 34th Chinese Control Conference (CCC)*,Hangzhou,2015,pp.4868-4872.
doi: 10.1109/ChiCC.2015.7260394

5. W. Pannao and C. Pintavirooj, "Application of direct linear transform for calibration of miniature computed tomography," *The 5th 2012 Biomedical Engineering International Conference*, Ubon Ratchathani,2012,pp1-5.
doi: 10.1109/BMEiCon.2012.6465468

6. F. Vigueras, A. Hernández and I. Maldonado, "Iterative Linear Solution of the Perspective n-Point Problem Using Unbiased Statistics," *2009 Eighth Mexican International Conference on Artificial Intelligence*,Guanajuato,2009,pp.59-64.
doi: 10.1109/MICAI.2009.39

7. L. D'Alfonso, E. Garone, P. Muraca and P. Pugliese, "P3P and P2P problems with known camera and object vertical directions," *21st Mediterranean Conference on Control and Automation*, Chania, 2013, pp. 444-451.
doi: 10.1109/MED.2013.6608760

8. J. Funda and R. P. Paul, "A comparison of transforms and quaternions in robotics," *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, USA, 1988, pp. 886-891 vol.2.
doi: 10.1109/ROBOT.1988.12172

9. E. Marchand, "VISP: a software environment for eye-in-hand visual servoing," *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, Detroit, MI, USA, 1999, pp. 3224-3229 vol.4. doi: 10.1109/ROBOT.1999.774089