Scientific Experimentation and Evaluation

Anirudh N J

Somesh Devagekar

Raina Thomas

June 26, 2019

Contents

1 Task 1 : Manual Motion Observation

1.1 Aim

The aim of this project is to construct a LEGO NXT differential drive robot and measure the observable end pose variation for three different trajectories: an arc to the left, driving straight and an arc to the right.

The experiment of measuring the end pose was done by measuring the pose using aruco markers and Computer Vision.

1.1.1 Experimental Setup

1.1.2 Software Setup

Software Installation

The following softwares were installed in the PC for driving the robot :

1. Java SE Development Kit 8 was installed in the Windows Operating System of the PC. The software was downloaded from the website : https://www.oracle.com

2. Eclipse IDE for Java Developers was downloaded and installed from the website http://eclipse.bluemix.net. This IDE is the platform used for developing the program for driving the robot.

3. The leJOS NXT Fantom Driver was downloaded from https://www.lego.com/en-us/mindstorms/download.

4. The leJOS NXJ software was downloaded from https://sourceforge.net and installed in the PC using the Windows Installer.

5. The leJOS NXJ plug-in was added to Eclipse.

Programming the robot

1. The Lego NXT Java template code provided in the website below was modified for use in the robot. https://lea.hochschule-bonn-rhein-sieg.de

2. A LeJOS NXT Project was created in Eclipse and the template code was added to the project and certain parameters in the code was modified. The parameters that were modified for our experimental setup are given in the table below.

Parameter	Value	Reason
SLEEP_TIME	3000.0	Increased to make the robot drive for a longer duration
TURN_RATE	10.0	Decreased to prevent the robot from making sharp turns

Table 1: Modified Code Parameters

3. The NXT Device was connected to the PC via a USB Cable. The NXJ firmware was flashed into the device. The modified code was compiled and run as an LeJOS NXT Program to create a .nxj file. This was automatically flashed into the NXT device that was connected to the PC.

1.1.3 Hardware Setup

To reproduce the experiment the following equipments are required :

1. White A0 Grid Sheet
2. Camera - Microsoft
3. Assembled robot - Lego NXT
4. Scales for measuring the length
5. Vision markers

The general steps to setup the experiment are as following :

1. Place the White A0 sheet on a flat surface. Try to remove any bends in the sheet and keep the paper as flat as possible. This will be called the 'map' on which the robot will move on. The measurement system is made of the robot, scale, camera, aruco markers and a paper.
2. Draw the template drawing of the "floor space" taken by the robot as shown in the figure. Concept of 'Behavior-shaping constraint'[1] is used to prevent wrong initial placement of the robot.

[1]Behavior shaping constraint : It is a type of design constraint to prevent the user from using the system in a way that it is not intended to be used in. In the case of the template given, the triangle prevents the robot to be placed in the reverse direction.

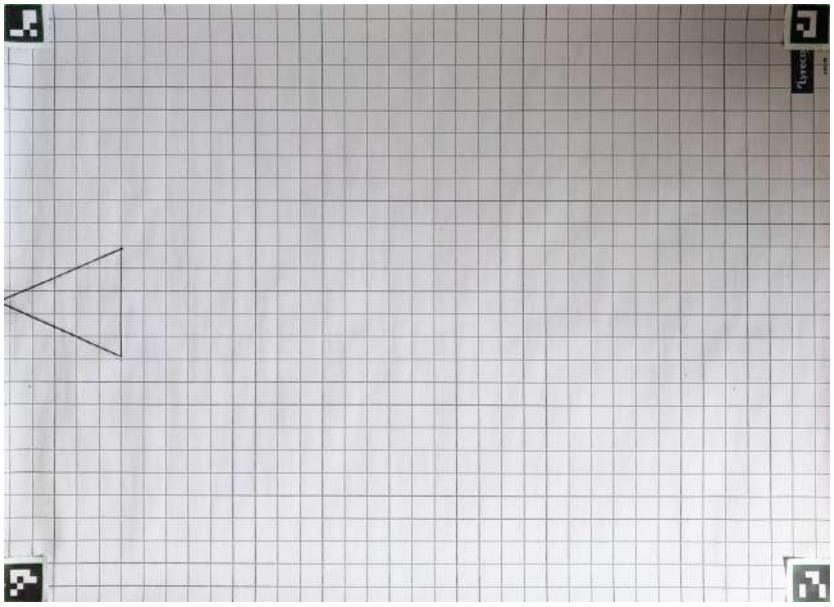


Figure 1: Template of the map

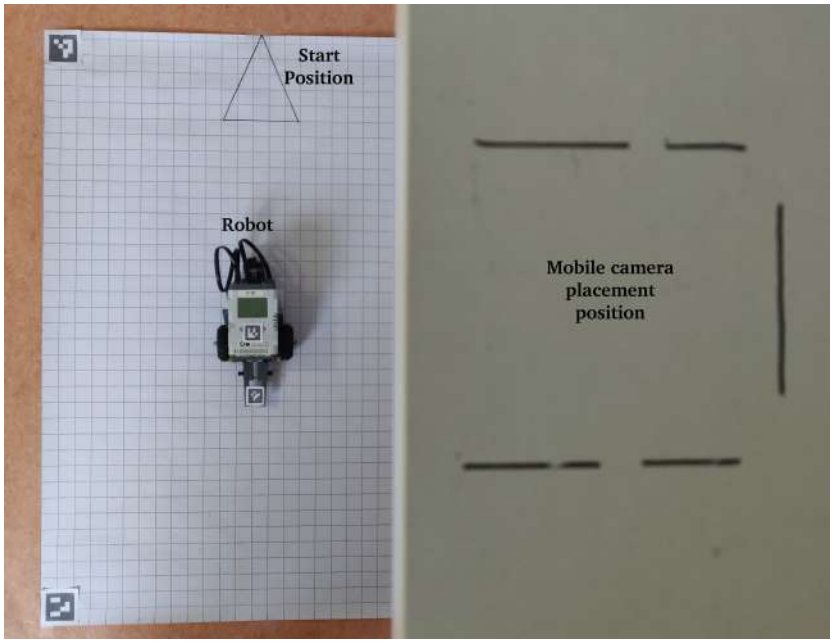


Figure 2: Top view of the experimental setup without camera

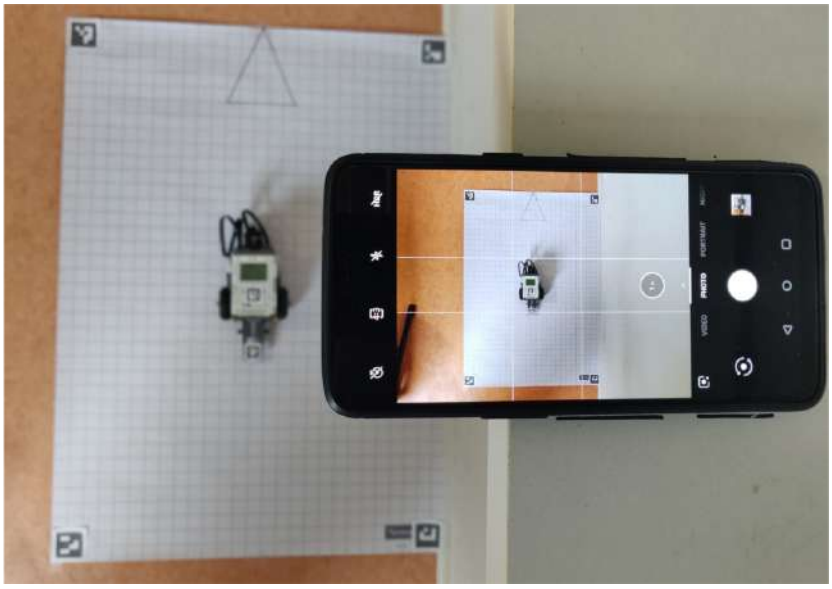


Figure 3: Top view of the experimental setup

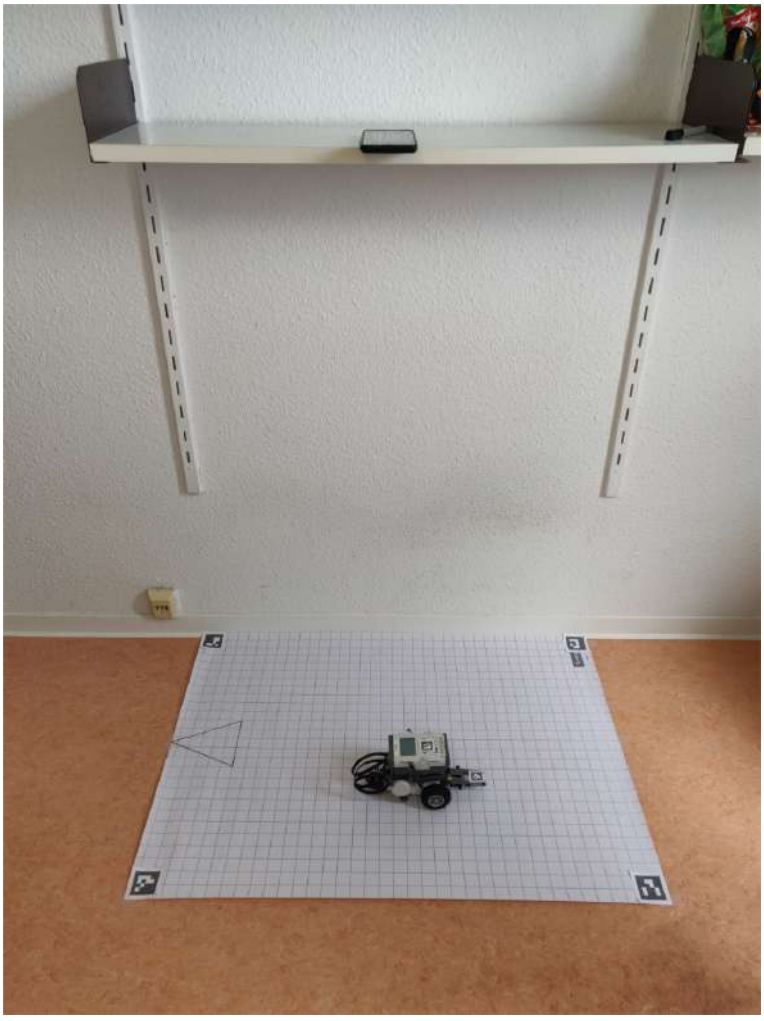


Figure 4: Front view of the experimental setup

3. Four aruco markers of different ids are used to find the corners of the grid sheet. The aruco markers are aligned such that the first corner of the marker which is detected by the CV algorithm overlaps with the corner of the sheet.
4. The markers are also aligned to be perfectly parallel to the sheet edges. These markers should be aligned properly since the next preprocessing steps are dependant on this.
5. The camera is placed rigidly such that the whole sheet is visible and the centre of the camera points approximately to the centre of the sheet.
6. The robot was assembled using the manual given with the LEGO NXT box. The robot firmware was flashed as given in the Software Setup.
7. We assumed that the driving axle is towards the front(leading edge) of the robot and that the robot origin lies in the centre of the driving axle.
8. The two aruco markers on the robot are placed as ahown in the figure in such a way that one of the markers lies directly on top of the origin of the robot.
9. Place the robot on the A0 sheet over the marked template in such a way that the castor wheel is turned outwards from the robot.
10. The measurands are the coordinates of the robot's markers on the camera.

The reference images of the DUT and the proposed marker tracking visualization are given below.

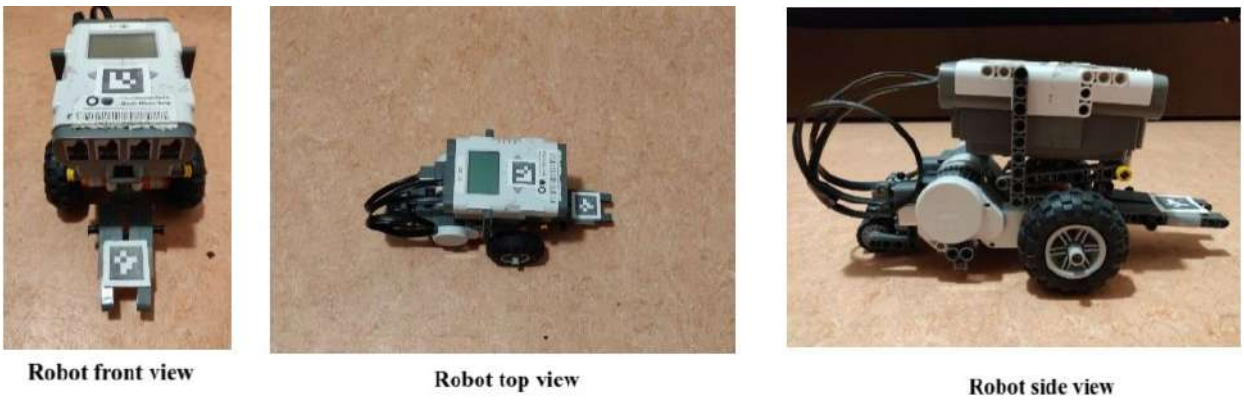


Figure 5: Views of the DUT

1.1.4 Camera Calibration

Calibration in computer vision is the method to find the intrinsic and extrinsic parameters of a particular camera. The intrinsic parameters of a camera deals with the

physical qualities of the camera like focal length , skew , distorsion etc. Whereas, the extrinsic parameters of the camera deals with finding the pose of the external world with respect to the camera.

The general methods to perform the calibration are as follows.

1. Download the image of the checkerboard pattern from the link given below. https://boofcv.org/images/2/23/Calibration_letter_chessboard_7x5.png
2. Take a printout of the pattern on an A4 sheet and mount it on a rigid surface. This will be called as the target.
3. Use the camera to be used in the setup to take pictures of the target at different orientation and distances.
4. Use the OpenCV code given in the below link to get the calibration parameters for the camera. https://docs.opencv.org/3.4.3/dc/dbb/tutorial_py_calibration.html
5. The calibration parameters that are found will be used for removing the errors like barrel, radial distorsion and also to find the precise 3D pose of the marker in the image.

1.2 General Terms Formalization

- **Measurand :**
  - Pose (x-coordinate, y-coordinate, orientation)
- **Measurement :**
  - Distance (mm)
  - Angle (degrees)
- **Measuring Principle :**
  - Computer Vision
- **Measurement facility :**
  - Camera
- **Device Under Test (DUT):**
  - Lego NXT
- **Measurement System :**
  - Grid Sheet

  - Markers
  - Camera

- **Measuring method:**
  - Run the robot on the grid sheet as explained in the next section.
  - Capture images using a fixed camera of the entire sheet at the starting position and ending position of the robot.
  - Find the pose of the robot by using the Computer Vision algorithm on the images captured.

1.3 Procedure

1.3.1 Experimental Steps

1. For this experiment, the camera should be placed rigidly directly above the experiment setup such that the full map is visible in the camera frame.
2. Before beginning the experiment the camera should be calibrated and the obtained parameters should be used to compensate for the lens distortion and other errors in the image.
3. Two small markers should be attached on the flat surface of the robot, one on the front and one on top of the origin of the robot. The markers should be visible in the camera frame captured by the overhead camera. (The two markers should have different ID to distinguish between them)
4. Using Computer-vision the two markers on the robot can be tracked and localized. The centre of the markers can be found and be used to calculate the offset from the actual robot origin using a scale or any other measuring device.
5. The robot is programmed to move using the initial pose and the desired final pose.
6. The initial pose and the final pose can be found by CV and position and pose in the required form can be extracted.
7. 20 measurements would be performed for each end point of the trajectory. Mean and standard deviation will be computed from the measurement. These measured values would be used to compute the accuracy and precision of the pose variation.

1.3.2 Computer Vision Algorithm

1. Preprocessing the image involves the following steps :
   (a) Camera calibration is done to find the intrinsic and extrinsic parameters of the camera. These parameters are then used to remove the radial distortion from the image.
   (b) The aruco markers on the four corners are then used to perform perspective transformation. This helps in removing minor skews in the image.
   (c) The four markers on the corners are then used to crop the image.

2. Using the preprocessed image, the algorithm tries to find the two aruco markers that are placed on the robot.
3. Using geometric calculations, the center of the aruco markers on the robots are found.
4. The marker positions are in pixels since it is directly calculated from the image. A transformation is performed on the pixel coordinates to convert it into 'millimeter' coordinates of the map.
5. The scripts written for the experiment along with the documentation of each of the function are given in the following repository. (https://github.com/njanirudh/SEE-Project)

Figure 6: Example of Aruco marker used for computer vision

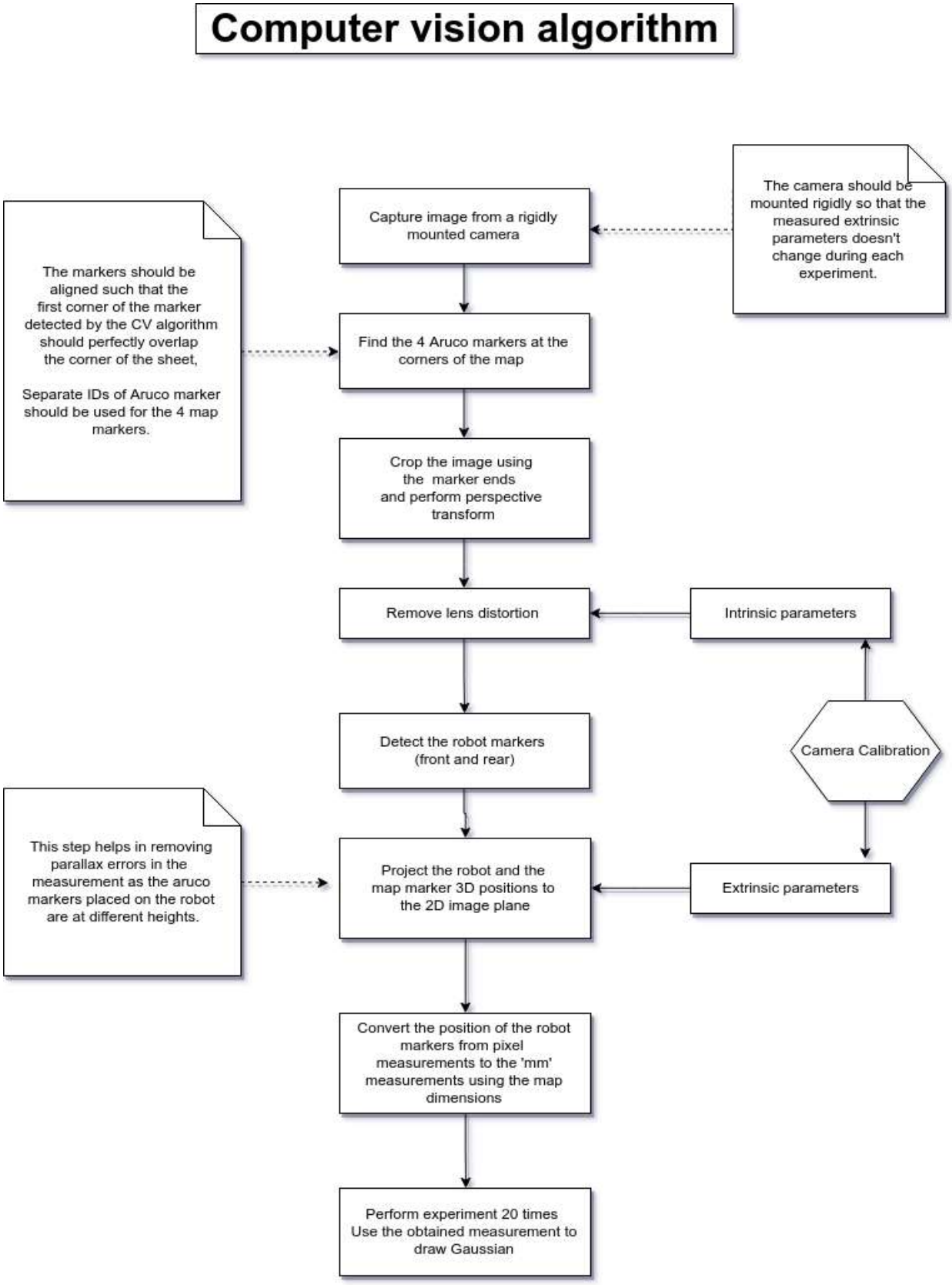


Figure 7: Algorithm Pipeline

1.4 Observations

1.4.1 Problems

1. This experiment accuracy and precision depended on the following aspects:
   (a) Camera Setup with respect to the grid sheet
   (b) Lighting conditions
   (c) Image recognition algorithms
2. The robot deviated from the expected trajectory towards the left due to the following reasons:
   (a) Uneven weight distributions of the robot
   (b) Difference in the tractions
   (c) Faster rotation of one motor compared to the other
3. Difference in the starting position of the robot during the experiment caused variations in the end pose of the robot.
4. Mechanical adjustments performed on the DUT during the experiment affected the end poses.
5. Pressing the start button of the robot at its start position also caused slight changes in the start pose.
6. Errors that occured due to the markers are given below:
   (a) A major source of error in the case of this experiment is that the compensation for parallax error when considering the markers is not done. This might cause the actual position of the marker corners to be slightly deviated from the result given without the parallax compensation.
   (b) The marker placed at the front of the robot gets hidden at times when the robot moves to the left. This prevents the camera from detecting the marker.
   (c) The bounding box may not encompass the whole marker. This causes inaccurate calculation of marker centers.

Figure 8: Hidden marker

Figure 9: Bounding Box Error

(d) The marker in the image appears skewed, which results in inaccurate calculation of pose.

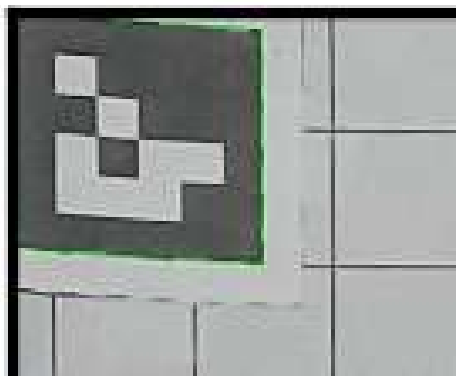


Figure 10: Skewed Marker

1.4.2 Data Visualisation

The origin of the original map (grid sheet) that was considered was at the top-left corner. However, the plots that have been displayed in this section have the origin in the bottom-left corner and hence the original images have been mirrored to fit the plot axes.

1. The scatter plot of the initial pose of the robot has been displayed in the figure below.

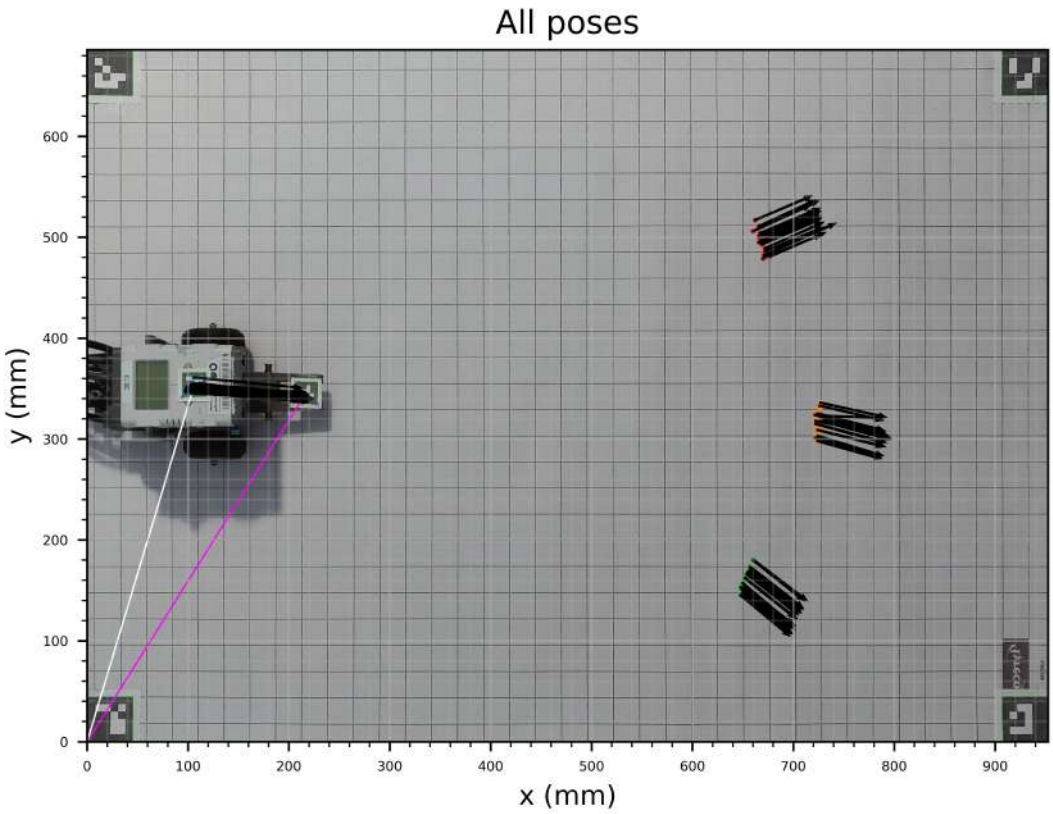


Figure 11: Initial Pose

2. The scatter plot of the end pose of the robot after it moved forward has been displayed in the figure below. It can be observed that there is a slight deviation of the robot on one side. This can be attributed to the mechanical faults in the robot or uneven weight distribution.

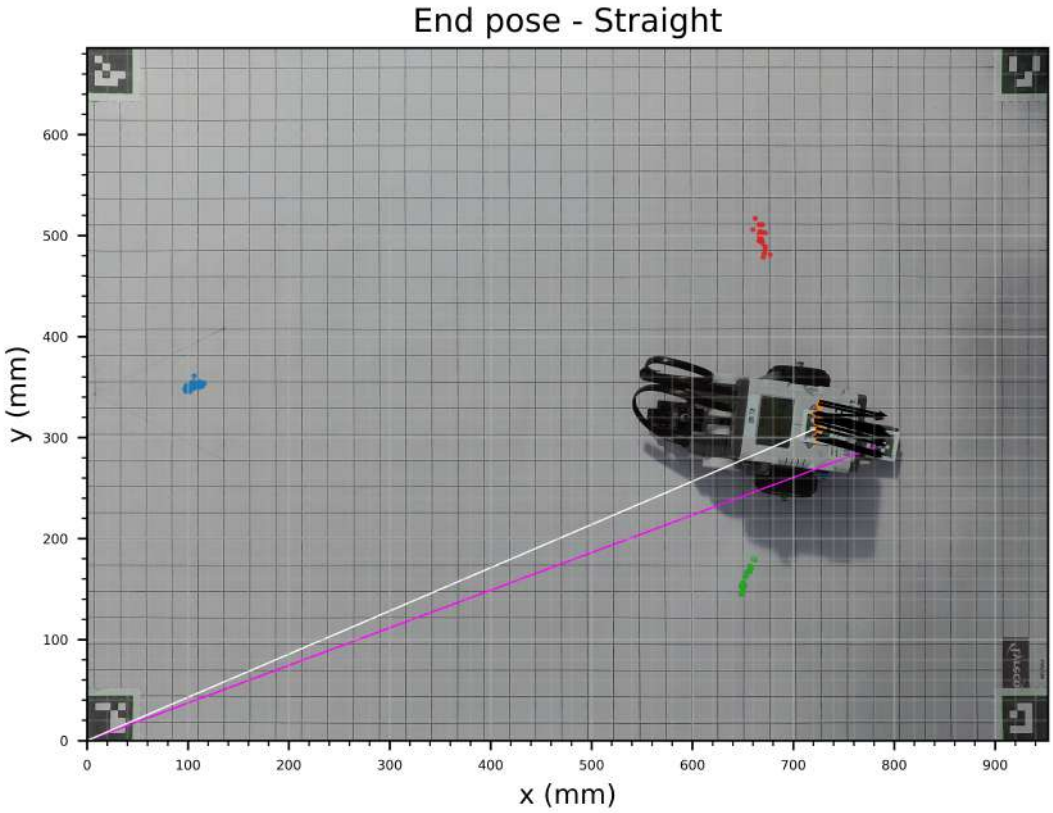


Figure 12: End Pose - Forward

3. The scatter plot of the end pose of the robot after it moved in the left arc has been displayed in the figure below.

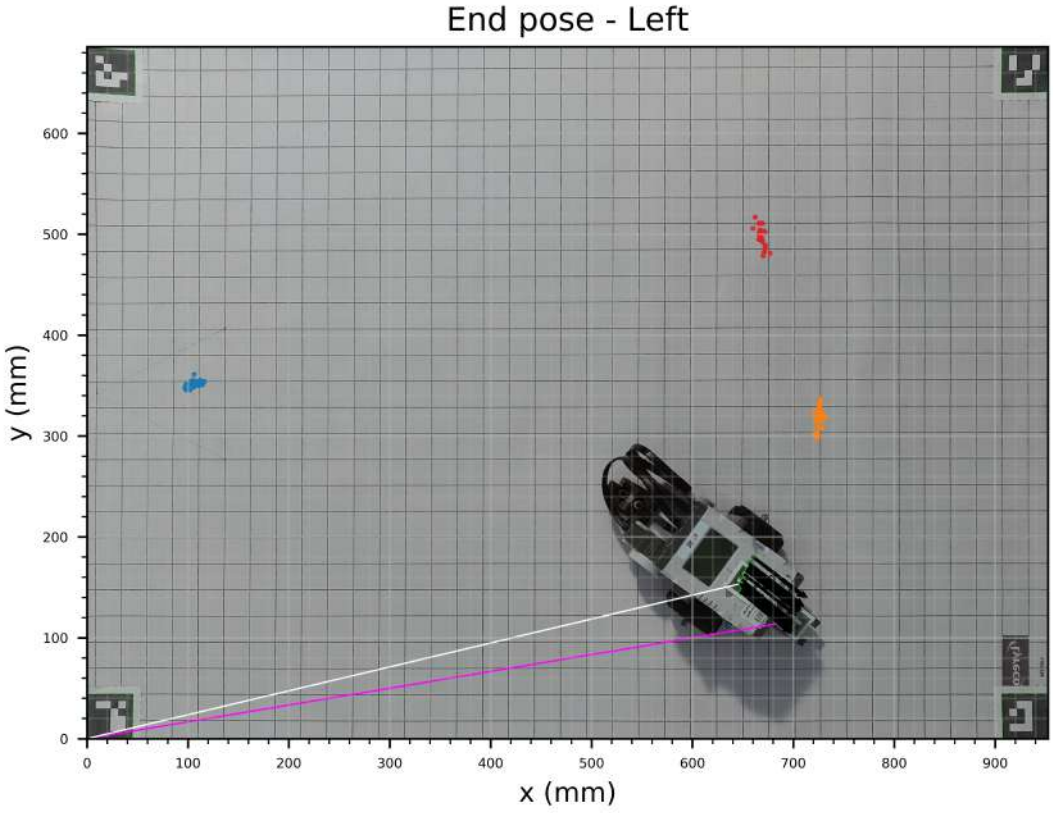


Figure 13: End Pose - Left

4. The scatter plot of the end pose of the robot after it moved in the right arc has been displayed in the figure below.

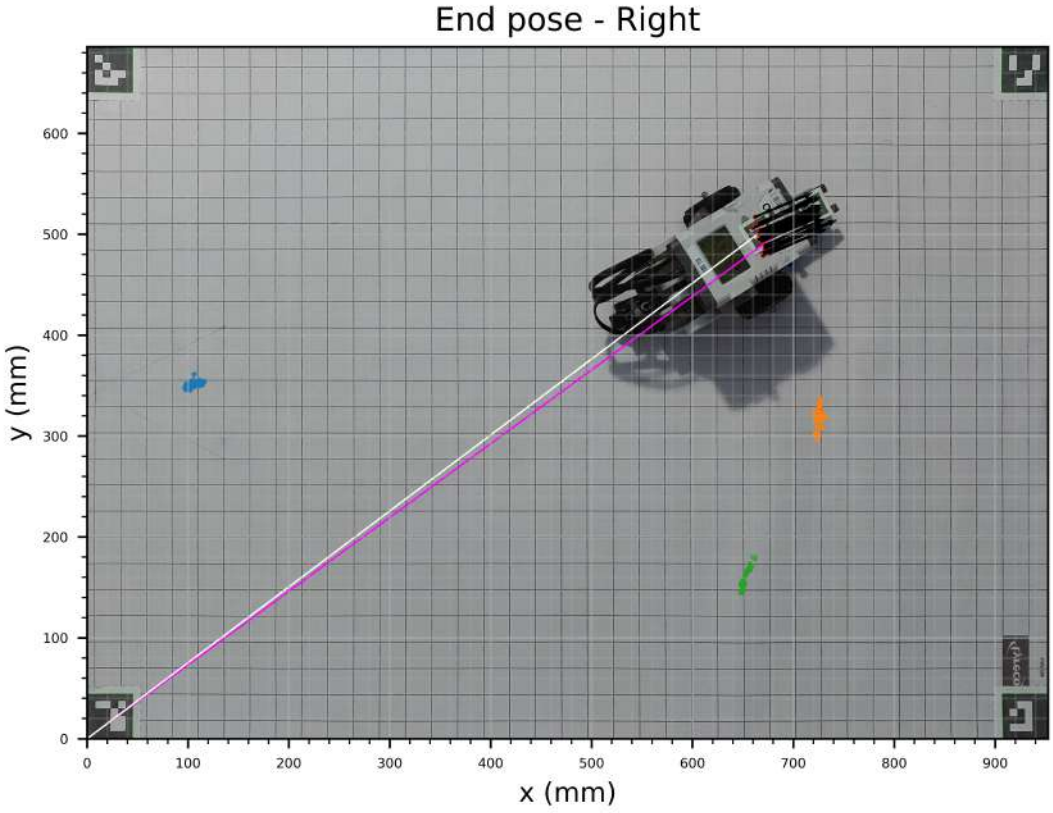


Figure 14: End Pose - Right

5. The plot of the poses of the robot has been displayed in the figure below.

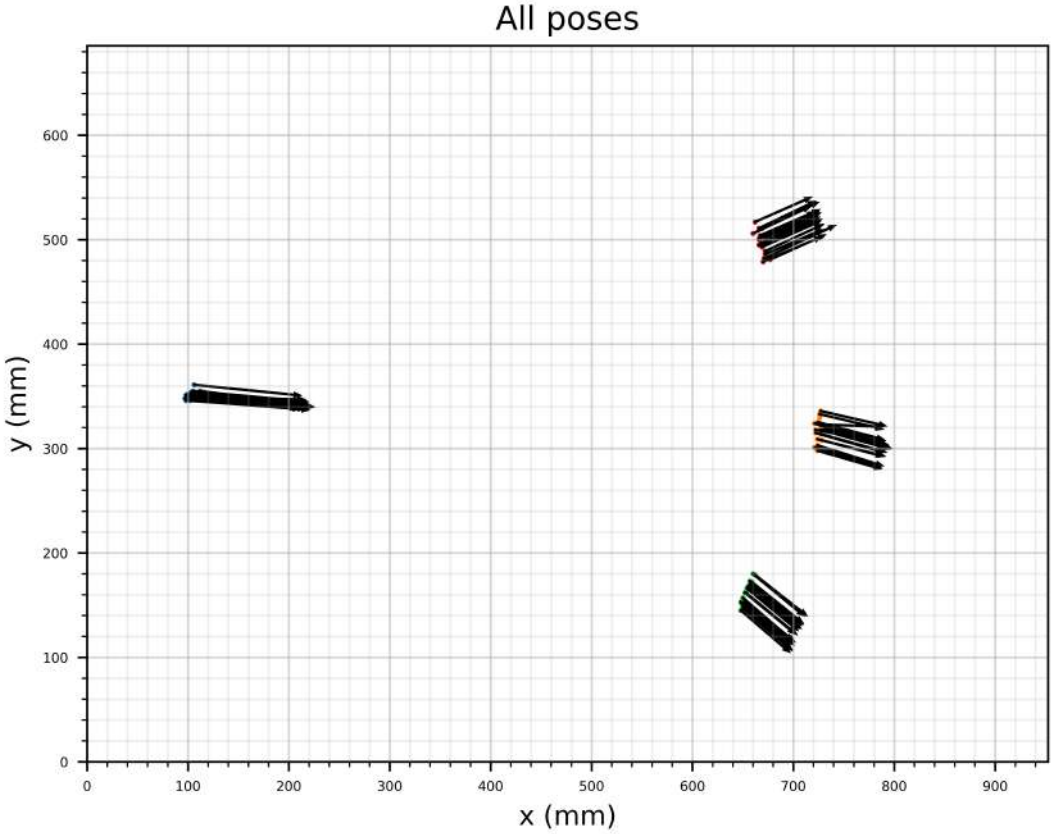


Figure 15: All Poses

2 Task 2 : Statistical Evaluation

2.1 Aim

The objective of this task is to characterise the observed behaviour from the previous experiment in terms of statistical parameters and estimate the distribution governing the spread of the stop positions.

2.2 Procedure

The data obtained from the previous experiment is used to fit to a Gaussian probability distribution by executing an algorithm in Python. The code used for this can be found in the github repository https://github.com/njanirudh/SEE-Project/tree/dev The general steps for statistical evaluating data are :-

1. The extreme outliers in the data are removed by using the Chebyshev's outlier algorithms.
2. Perform PCA on the x,y and the angle data for each of the three poses (start, forward, left and right). PCA will help in mean-centering the data while selecting the axis where the projection of the actual data is maximum.
3. Chi-squared test is performed on the data to check if the data fits a gaussian distribution or not. Since Chi-squared test requires a minimum of 5 points, the number of bins selected for 20 observations is 5.
4. Python functions provided in the github repository are used to perform PCA of the data, chi-squared test and also generate the histograms given below.

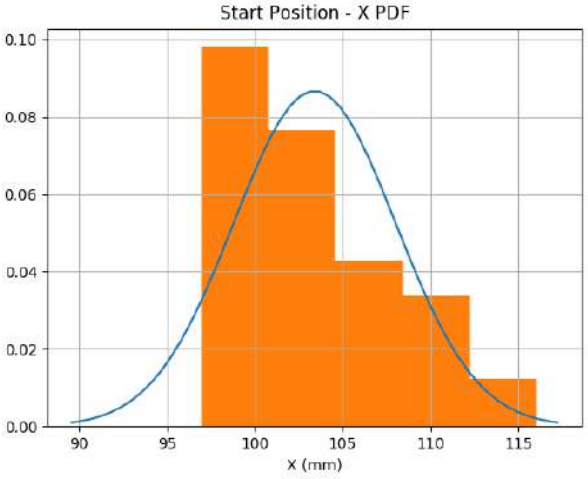
2.3 Algorithms

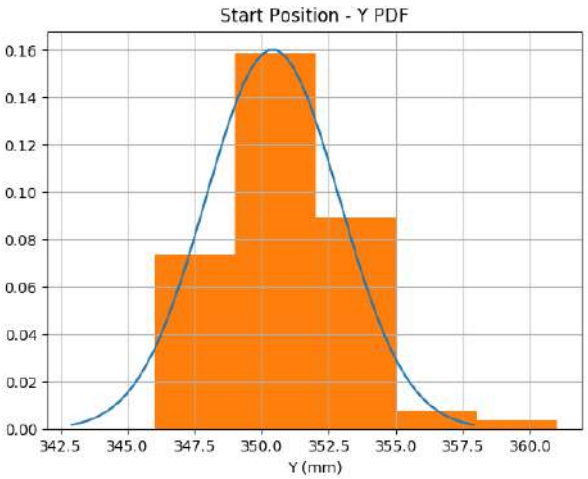
2.3.1 Outliers Detection

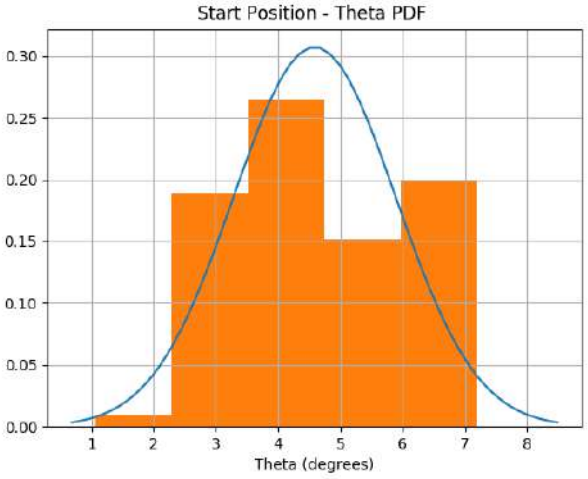
```
def remove_outliers(data, pp1, pp2):
"""
Based on "Data Outlier Detection using the Chebyshev Theorem",
Brett G. Amidan, Thomas A. Ferryman, and Scott K. Cooley
Keyword arguments:
data -- A numpy array of discrete or continuous data
pp1 -- likelihood of expected outliers (e.g. 0.1, 0.05 , 0.01)
pp2 -- final likelihood of real outliers (e.g. 0.01, 0.001 , 0.0001)
"""
mu1 = np.mean(data)
sigma1 = np.std(data)
k = 1./ np.sqrt(pp1)
odv1u = mu1 + k * sigma1
odv1l = mu1 - k * sigma1
new_data = data[np.where(data <= odv1u)[0]]
new_data = new_data[np.where(new_data >= odv1l)[0]]
mu2 = np.mean(new_data)
sigma2 = np.std(new_data)
k = 1./ np.sqrt(pp2)
odv2u = mu2 + k * sigma2
odv2l = mu2 - k * sigma2
final_data = new_data[np.where(new_data <= odv2u)[0]]
final_data = new_data[np.where(final_data >= odv2l)[0]]

return final_data
```

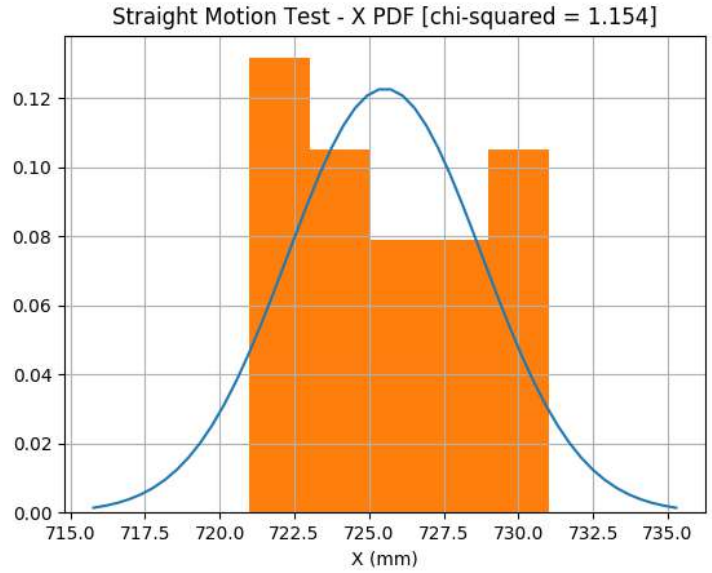
2.4 Observations

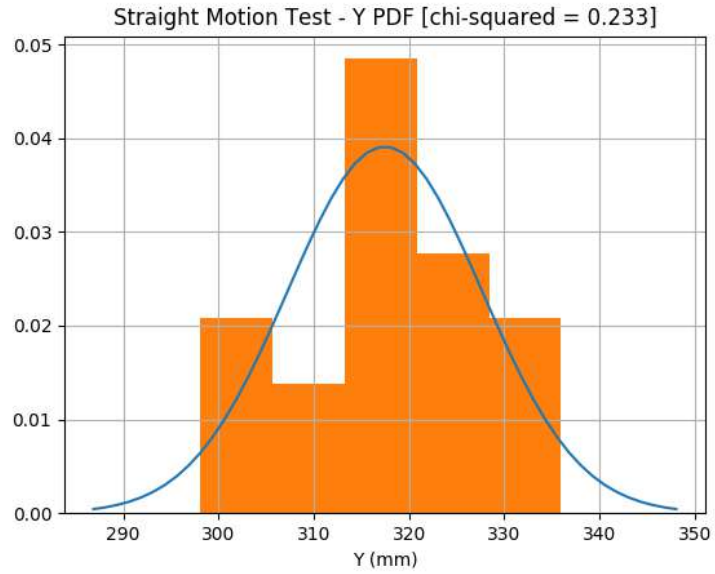
Given below are the curves for Gaussian fit from the observed data along with values from the chi-squared test without PCA.

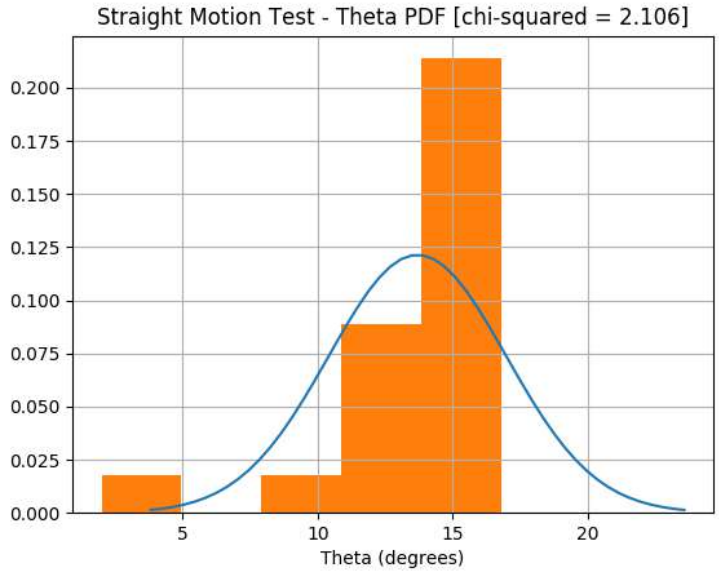


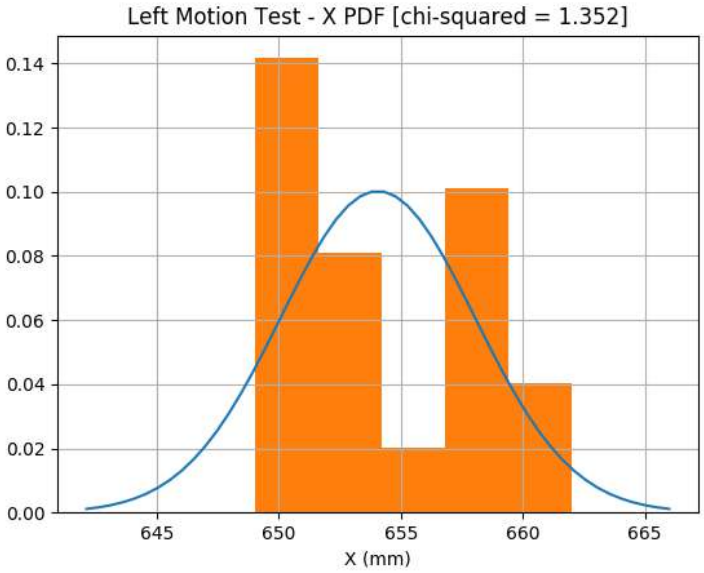


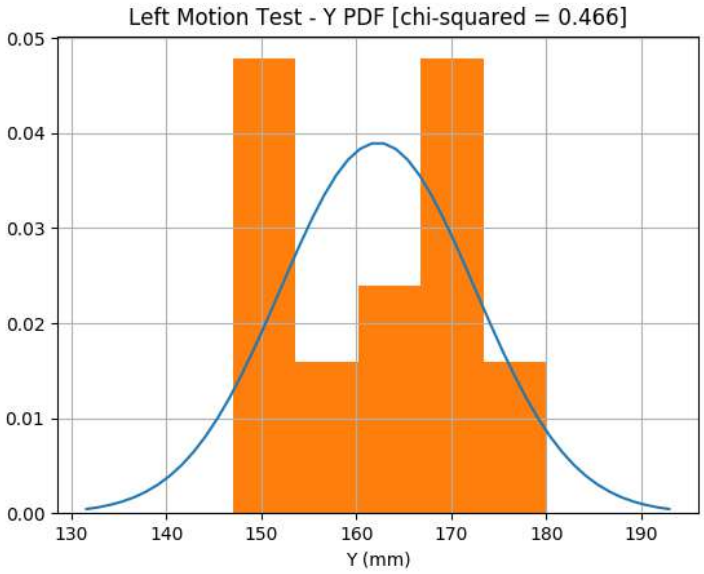


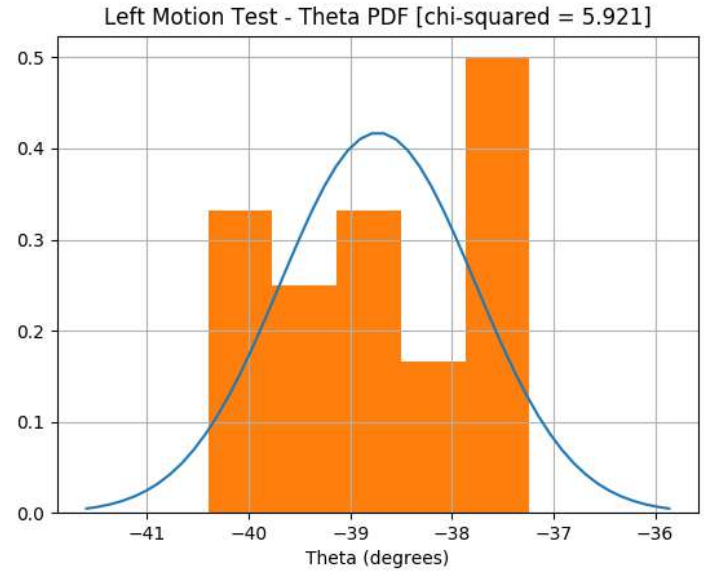

Straight Motion Test - X PDF [chi-squared = 1.154]
0.12
0.10
0.08
0.06
0.04
0.02
0.00
715.0
717.5
720.0
722.5
725.0
727.5
730.0
732.5
735.0
X (mm)

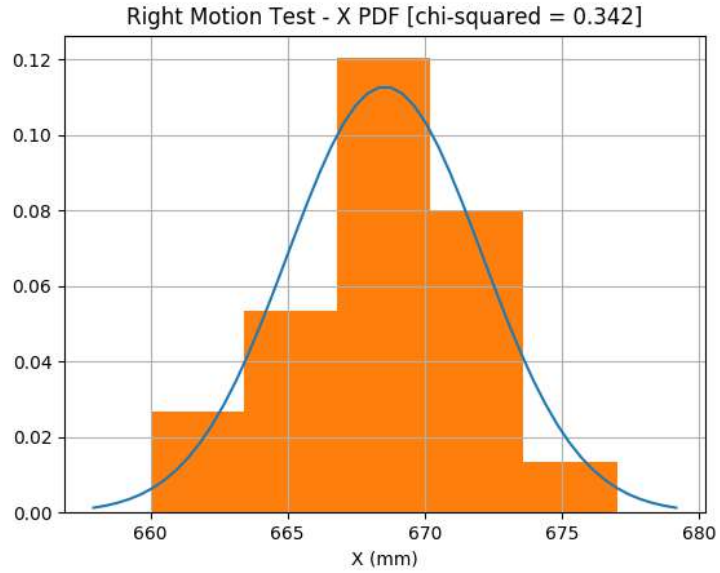

Straight Motion Test - Y PDF [chi-squared = 0.233]
0.05
0.04
0.03
0.02
0.01
0.00
290
300
310
320
330
340
350
Y (mm)

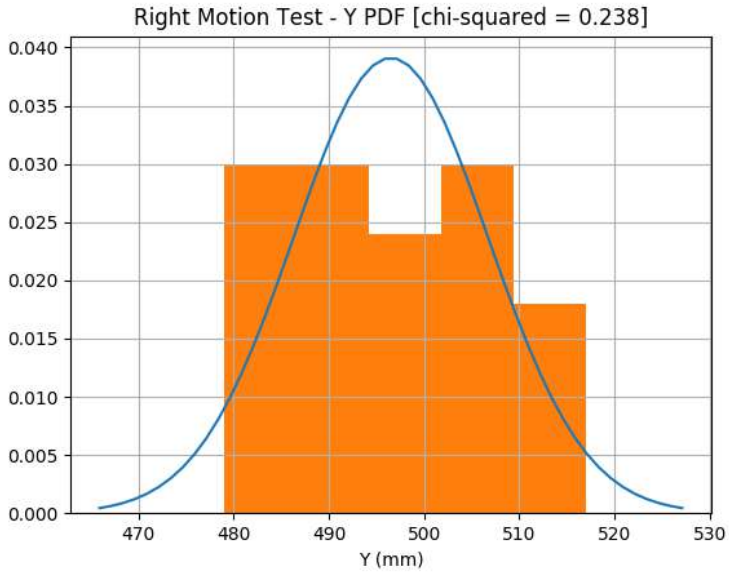

Straight Motion Test - Theta PDF [chi-squared = 2.106]
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
5
10
15
20
Theta (degrees)

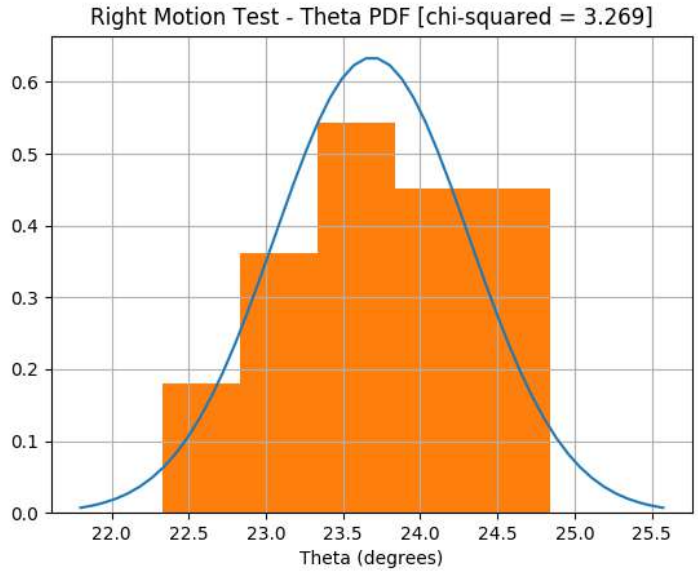

Left Motion Test - X PDF [chi-squared = 1.352]
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
645
650
655
660
665
X (mm)


Left Motion Test - Y PDF [chi-squared = 0.466]
0.05
0.04
0.03
0.02
0.01
0.00
130
140
150
160
170
180
190
Y (mm)

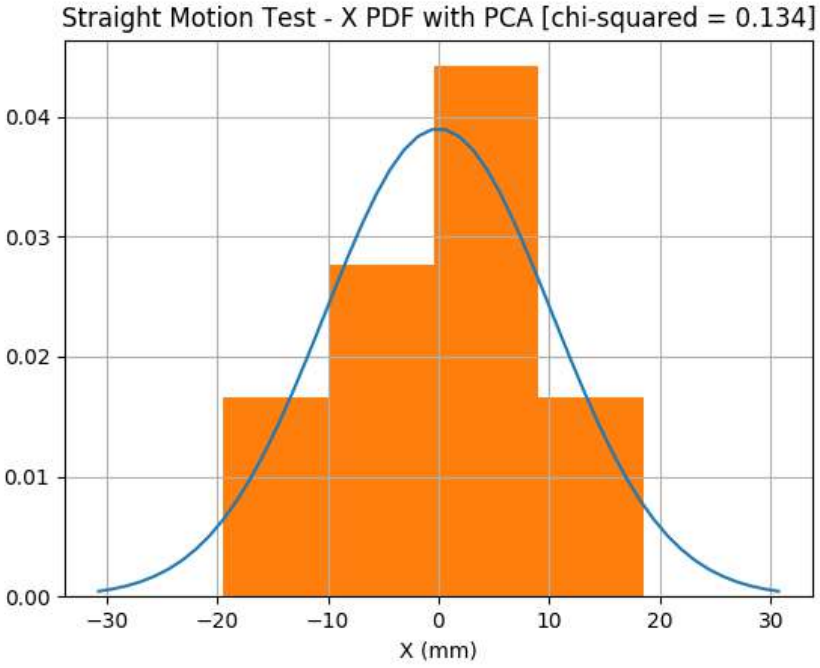

Left Motion Test - Theta PDF [chi-squared = 5.921]
0.5
0.4
0.3
0.2
0.1
0.0
−41
−40
−39
−38
−37
−36
Theta (degrees)

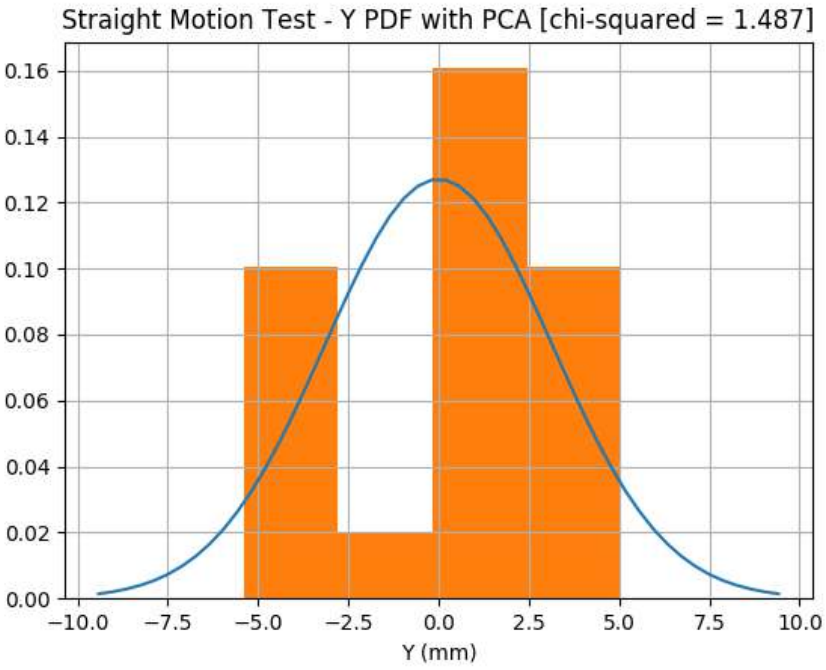

Right Motion Test - X PDF [chi-squared = 0.342]
0.12
0.10
0.08
0.06
0.04
0.02
0.00
660
665
670
675
680
X (mm)

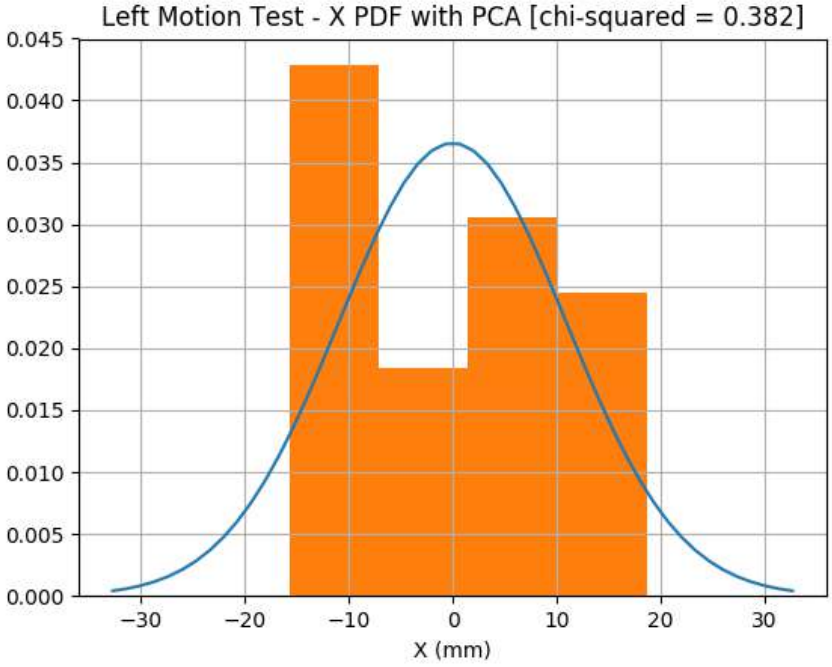

Right Motion Test - Y PDF [chi-squared = 0.238]
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
470
480
490
500
510
520
530
Y (mm)

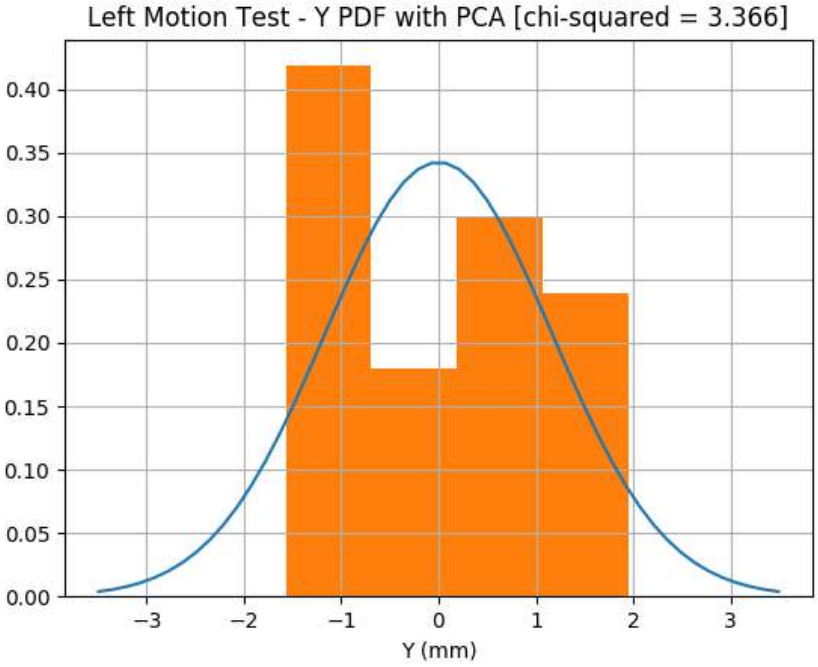

Right Motion Test - Theta PDF [chi-squared = 3.269]
0.6
0.5
0.4
0.3
0.2
0.1
0.0
22.0
22.5
23.0
23.5
24.0
24.5
25.0
25.5
Theta (degrees)

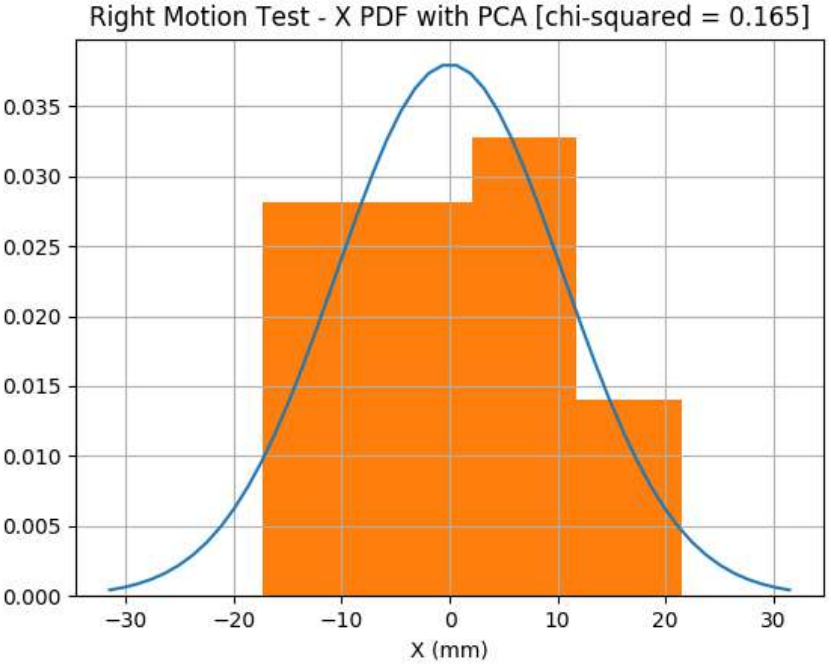
Given below are the curves for Gaussian fit from the observed data along with values from the chi-squared test with PCA.

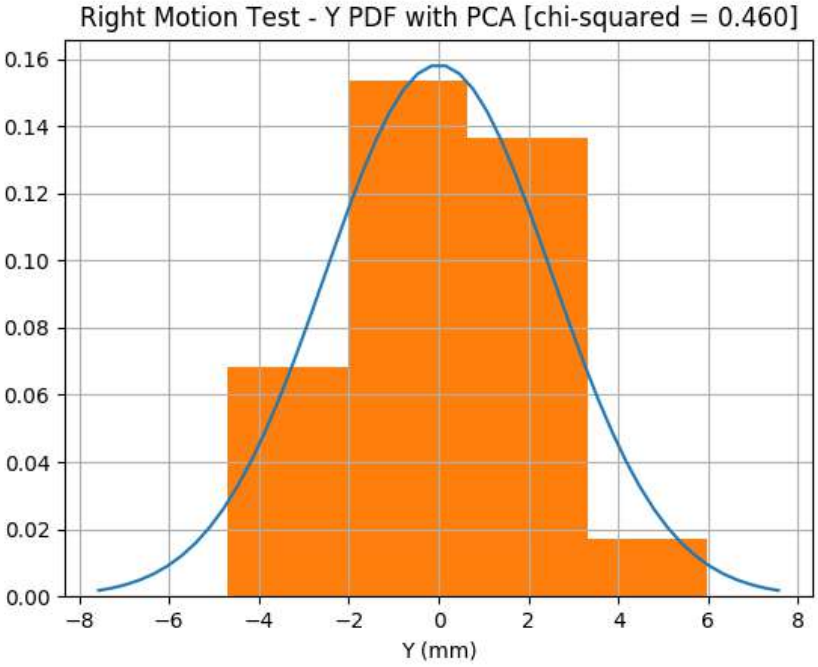


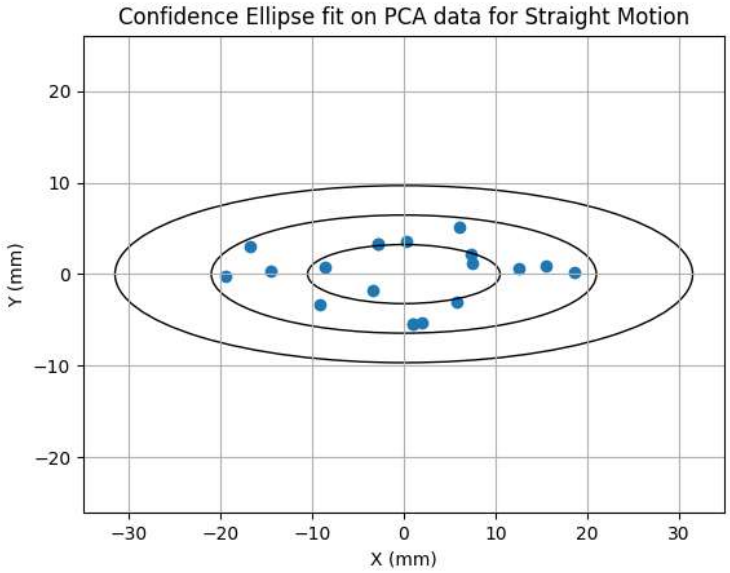


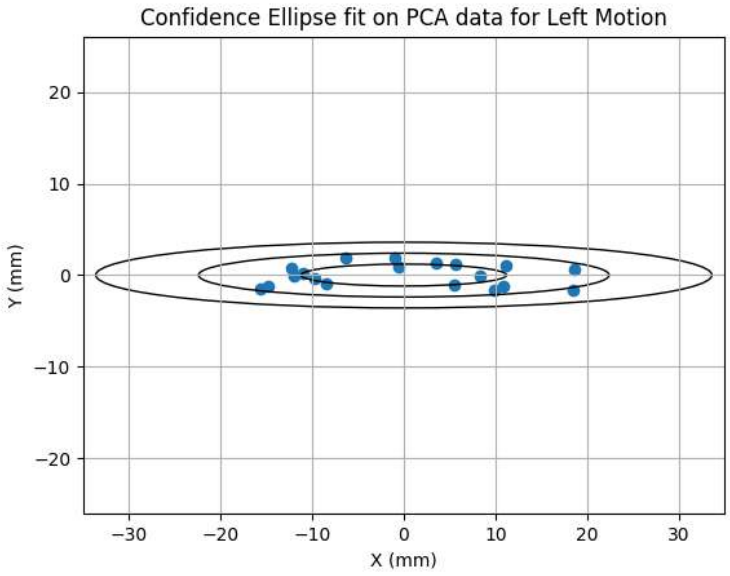

Left Motion Test - X PDF with PCA [chi-squared = 0.382]
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
−30
−20
−10
0
10
20
30
X (mm)

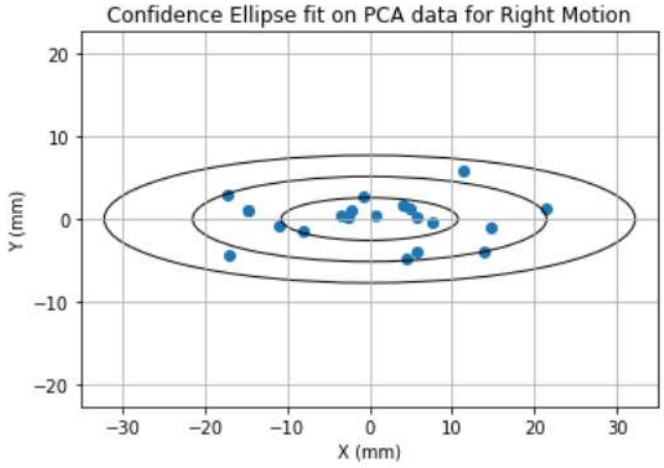

Left Motion Test - Y PDF with PCA [chi-squared = 3.366]
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−3
−2
−1
0
1
2
3
Y (mm)


Right Motion Test - X PDF with PCA [chi-squared = 0.165]
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
−30
−20
−10
0
10
20
30
X (mm)


Right Motion Test - Y PDF with PCA [chi-squared = 0.460]
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−8
−6
−4
−2
0
2
4
6
8
Y (mm)







The data obtained from the experiments performed can be used to measure the

precision and accuracy.

2.4.1 Accuracy

To calculate the accuracy, we need to understand the ground true value(true pose) and the actual value (achieved pose). Accuracy is the amount of deviation from ground truth to achieved result. In our experiment, accuracy is the deviation of the actual pose to the true pose.
Calculation of actual pose is given by the following:
Given:
SPEED=180 mm/s
SLEEP_TIME= 3 s
TURN_RATE= 10%

The robot has three end poses accordingly straight, right and left end pose. The true straight pose is given by $d = SPEED \times TIME$

$\implies d = 180 * 3 = 540mm$

Obtaining left or right true pose:
Since the robot is a differential drive robot, varying the velocity of each wheel will make the robot to perform rotation about a point called the instantaneous center of rotation(ICC).
Given velocity of each wheel (V_r, V_l) and distance between two wheels (l).

$$V_r = 180mm/s$$

$$V_l = \frac{180}{100} \times (180 - 10) = 162mm/s$$

$$l = 120mm$$

To have (R) which is the distance from the ICC to the mid-point between the wheels(l)

$$R = [\frac{l}{2}] \times [\frac{V_l + V_r}{V_r - V_l}]$$

$$l = 120mm$$

$$R = [\frac{120}{2}] \times [\frac{162 + 180}{180 - 162}] = 1140mm$$

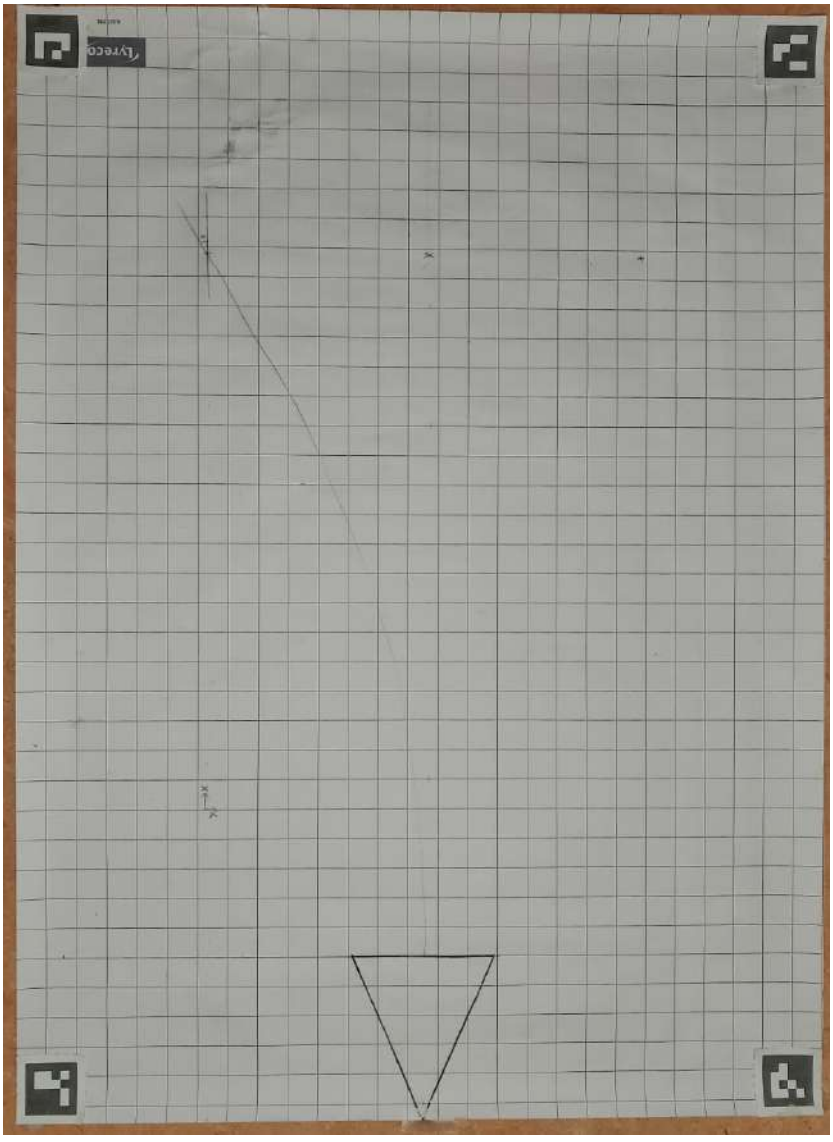
For the rotation rate

$$\omega = \frac{V_r - V_l}{l}$$

$$\omega = \frac{180 - 162}{120} = 0.15 rad/s$$

Thus, for true right, d = $\omega \times SLEEP\_TIME = 0.15 \times 3 = 0.45$ rad= 25.78 deg
Using this ICC, we project the corresponding angle onto the grid sheet as show in the figure below. Since our marker is located in the center of the axle, the true pose is projected onto the sheet, using the following steps:

1. Place the robot onto to the designated origin position, as shown in the figure.
2. Accordingly from the position of the marker, measure the distance R with respect to the below figure.
3. Using basic geometric apparatus, draw a quadrant of a circle, such that it begins at the marker position and follows upto the rotation rate angle(d).
4. Resulting is the true end pose for right/left trajectories.



The true poses for the three trajectories are shown below.

True pose	Achieved mean pose	Difference
730	700.38	30.38
510	520.14	10.14
29	24.814	4.18

Figure 16: Right Arc Motion

True pose	Achieved mean pose	Difference
686	690.38	4.38
162	171.27	-9.27
-38	-40.88	2.88

Figure 17: Left Arc Motion

True pose	Achieved mean pose	Difference
720	765.83	45.83
302	335.11	33.11
0	14.45	14.45

Figure 18: Straight Motion

2.4.2 Precision

1. The observation in the table shows that the variance along y direction is more than along x direction when the robot moves in arc.
2. The above is attributed to the fact that a small change in the angle causes a bigger change in y direction, the further the robot drives. Thus also the precision in x direction is better than in y.

Parameter	Forward	Right	Left
Mean_X(mm)	765.83	700.38	690.38
Mean_Y(mm)	335.11	520.14	171.27
Mean_Theta(deg)	14.45	-24.81	40.88
Variance_X(mm)	± 10.56	±12.52	±15.83
Variance_Y(mm)	±104.03	±104.06	±104.61
Variance_Theta(deg)	±10.8	±0.39	±0.91

Table 2: Observed Parameters

3. It can also be seen that the distance in x direction for the right arc is bigger than for the left, while the distance in y is bigger for the left arc. An explanation for this could be that the right wheel is rotating faster than the left wheel. This is also supported by the fact that the robot moves slightly to the left when programmed to move straight.

2.5 Conclusion

An experiment was performed using LEGO NXT robot. 20 measurement of the pose were taken from a common start position and then after reaching an end position (straight, left and right). The measurements were plotted on the graph and it was concluded by performing a chi-squared test that the measurements follow a Gaussian distribution.

3 Task 3 : Camera Calibration

3.1 Aim

The goal of this report is to provide a general introduction about the the theory of camera calibration and provide the general steps to calibrate a Microsoft Lifecam HD camera using OpenCV. This report will conclude by giving the intrinsic and extrinsic camera parameters.

3.2 Introduction

Calibration of the camera is the method in computer vision to find the properties of the camera hardware. It is a major prerequisite for performing any type of measurement using camera or 3D projections and reconstructions.

Three specific kinds of camera calibration techniques exist :

1. *Photogrammetric calibration* : This technique makes use of a calibration object of specific size and geometric properties whose geometry in 3D space is very precisely known. This calibration object usually have 2-3 orthogonal planes.
2. *Self-calibration* : This method doesn't make use of any kind of calibration object. Camera displacement information from multiple images is used to get the calibration parameters for the camera. Although these methods are very flexible as no setup is required, but these methods are still not mature and cannot always give reliable results.
3. *Hybrid system* : This methods make use of simple planar patterns to find the calibration parameters. Unlike the photogrammetric methods, this method doesn't require any specific motion information and makes use of only the 2D metric information obtained from the image. Few types of planar calibration boards include checkerboard targets, circular targets, charuco targets etc. This method is also referred to as "Zhang's Camera Calibration Algorithm".

There are two types of distortions that are majorly found in the cameras :

1. *Radial distortion* : This types of distortion causes the straight lines present in the images to appear curved. This phenomenon is caused because the light gets bent more near the edges compared to the optical center. The smaller the lens, the more the radial distortion. These can be further divided into Barrel distortion and Pincushion distortion.

2. *Tangential distortion* : This type of distortion occurs when the image and the lens plane of the camera are not parallel. This also include an offset between the lens and image plane centre.

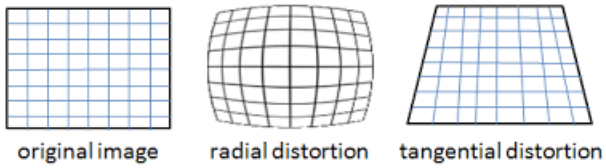


Figure 19: Types of distortions[2]

For this report, the OpenCV's implementation of Zhang's Algorithm is used to obtain the calibration characteristics. Apart from having a very simple setup, this method can be used to obtain precise enough values for both the intrinsic and the extrinsic parameters.

3.3 Calibration algorithm

Zhang's Algorithm make use of planar calibration targets to calculate the camera intrinsic and extrinsic parameters.

For many of the modern cameras, we consider the reduced models to find the calibration parameters as they give a fairly good estimates of the camera properties and are simple to calculate. The following are the assumptions that Matlab and OpenCV calibration functions consider when performing calibrations :

1. The algorithm is used to measure only the radial distortion not tangential distortion. This is because for webcams and other simple cameras the radial distortion is the major source of distortion. The tangential distortion of modern cameras are nearly negligible due to very less imperfections in manufacturing. Ie. 6th Order Calibration Models (Radial + Tangential) are not considered for the calculations by default in Matlab Camera Calibration Toolkit[1]. But OpenCV considers both the tangential and the radial distortions k_1, k_2, p_1, p_2, k_3 [2]
2. According to the Matlab Camera Calibration Toolbox documentation it is given "For standard field of views (non wide-angle cameras), it is often not necessary (and not recommended) to push the radial component of distortion model beyond the 4th order (i.e. keeping kc(5)=0)"[1]. But OpenCV considers even the 6th order component, also referred to as k_3, of the distortion model[2].
3. The modern camera pixels are rectangular in shape so the skew component in the calibration matrix is not considered [2][1]

4. There are no fixed number of images that must be used for the calibration. According to the paper by [3] it is given that "...the camera to observe a planar pattern from a few (at least two) different orientations". But OpenCV recommends atleast 10 images to get a very reasonably good calibration parameters.
5. For a fixed camera the intrinsic parameters can be reused for capturing multiple images. It is very important to fix the camera focus before capturing the images for calibration task.

3.4 Experimental setup

The following are the software and hardware requirements to perform the camera calibration:

3.4.1 Hardware

1. Microsoft Lifecam HD camera
2. Camera clamp
3. Checkerboard target printout
4. Rigid piece of cardboard

3.4.2 Software

1. Ubuntu OS
2. Python 3
3. OpenCV 3.X

The setup for the experiment is given below.

3.5 Procedure

1. Connect the camera clamp to a rigid table or a support.
2. Attach the camera rigidly to the clamp such that there is no movement in the setup when capturing image.
3. Take a high quality/high contrast printout of the checkerboard target from the link.
4. Stick the printed target to a rigid planar surface such as a solid cardboard. This will be further referred to as the *calibration target*.

Figure 20: Experimental Setup

5. The resolution of the camera is set at Full HD resolution (1920 x 1080).
6. Take many picture of different orientations of the calibration target from the camera setup. For this report, a total of 50 images were taken.
7. Use the images to find the camera extrinsic and intrinsic parameters using the OpenCV scripts.

3.6 Results

The obtained corners for the camera calibration are as following.

3.6.1 Intrinsic parameters

The camera matrix indicates the physical parameters of the camera like the focal lengths (f_x,f_y), the principal point coordinates (c_x,c_y) in 'mm'

$$\begin{vmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1471.25 & 0 & 889.90 \\ 0 & 1473.07 & 618.63 \\ 0 & 0 & 1 \end{vmatrix}$$

The ideal centre values of the camera frame is (960 x 540) as the camera has a resolution of (1920 x 1080) pixel.

3.6.2 Reprojection Error

Reprojection error is the distance between the projected point and the actual value. OpenCV uses the calculated extrinsic parameters to find the actual measurements and then calculates root mean square error to find the re-projection error. Reprojection error is usually expected to be within [0,1].

Reprojection error = 0.886

3.6.3 Distortion coefficients

Distortion coefficients = [k1 , k2 , p1 , p2 , k3] = [-3.9e-02 3.4e-01 5.9e-04 9.1e-04 -7.1e-01]

Multiplying the values given above with the length of the squares sides(70mm): The Units of the distortion coeff. are in 'mm'. Distortion coefficients = [k1 , k2 , p1 , p2 , k3] = [-27.3e-02 23.8e-01 41.3e-04 63.7e-04 -49.7e-01]

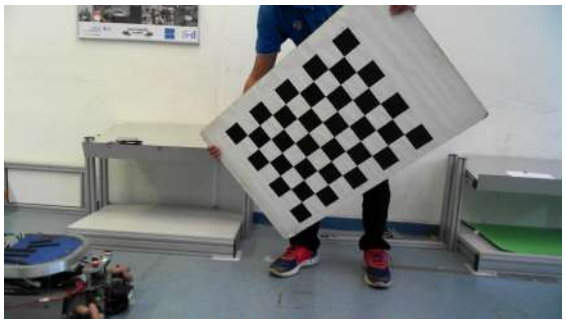
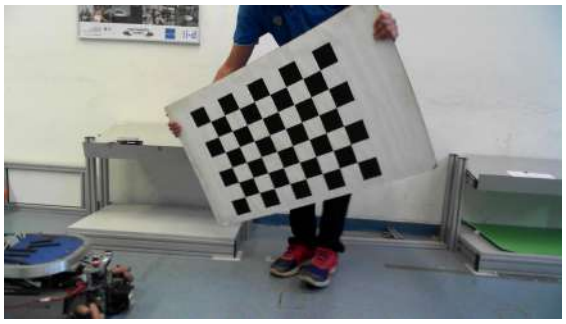
k1, k2, k3 are radial distortion coefficients. p1 and p2 are tangential distortion coefficients.

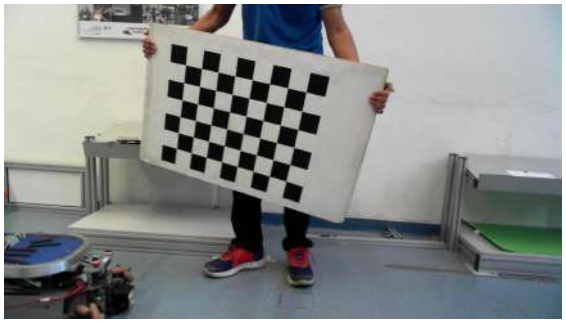
Calibration Images









3.6.4 Extrinsic parameters

The extrinsic parameters (rotation and the translation vector) of the marker top left corner is found wrt. the camera frame. The units of the 'tvec' is in millimeters (mm). The 'rvecs' is represented as an angle-axis, so the norm will have units radians, and the axis is unit-less.

$$\text{The rotation vector rvec} = \begin{vmatrix} 0.21 \\ 0.22 \\ -0.01 \end{vmatrix}$$

$$\text{The translation vector tvec} = \begin{vmatrix} 75.74 \\ 116.95 \\ 2156.39 \end{vmatrix}$$

3.7 Possible Errors

1. The auto-focus of the camera must be turned off before performing the experiment. For multi focal cameras, different intrinsic parameters exists for different focal lengths of the lens.
2. The image should be taken at the highest resolution possible for the camera.
3. The camera must be fixed rigidly and only the target should be moved.
4. The contrast of the pattern must be high enough so that it can be detected easily in the image. Corner point of the pattern must always be visible for the calibration algorithm to work properly.

3.8 Conclusions

Camera calibration parameters for a Microsoft Lifecam HD camera were estimated using the OpenCV.

3.9 Code

```python
import numpy as np
import cv2 as cv
import glob

# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*8,3), np.float32)
objp[:,:2] = np.mgrid[0:8,0:6].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

# Iterate through all Calibration images
images = glob.glob('DataSets/Calibration/*.jpg')
for i,fname in enumerate(images):
    img = cv.imread(fname)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # Find the chess board corners according to the size of the pattern
    # The pattern size according to this report is (8 x 6)
    ret, corners = cv.findChessboardCorners(gray, (8,6), None)

    # If pattern is  found, add object points,
    # image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        # Refining the corner found on the pattern in the calibration image
        corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)

        # Draw and display the corners
        cv.drawChessboardCorners(img, (8,6), corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(500)

cv.destroyAllWindows()

ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],
                                                  cv.CALIB_ZERO_TANGENT_DIST, None)
```

Figure 21: OpenCV code

References

[1] 2015. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/parameters.html

[2] 2018. [Online]. Available: https://docs.opencv.org/3.4.3/dc/dbb/tutorial_py_calibration.html

[3] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, pp. 1330–1334, 2000.

[4] W. Burger, "Zhangs camera calibration algorithm: In depth tutorial and implementation," no. HGB16-05, may 2016. [Online]. Available: http://staff.fh-hagenberg.at/burger/

[5] 2014. [Online]. Available: https://fivedots.coe.psu.ac.th/~ad/jg/nuiN13/depthProcessing.pdf

4 Task 4 : Measuring the Accuracy and Precision of a KUKA youBot Arm

4.1 Aim

The objective of this task is to estimate the accuracy and precision of a robotic arm. The experiment involves a robot performing different pick and place tasks; in particular,a robot (a KUKA youBot arm) will be placing objects with three different masses in three different poses (straight, left, right). The object positions (x, y) are determined by tracking ArUco markers that are placed on top of the objects; in particular, an external vision system (a camera) is placed above the arm's workspace in order to determine the marker poses.

4.2 Setup and Procedure

The experimental setup and procedure has been explained in detail in the *SEE_Manual For Practical Experiments : Section 2.4*

4.3 Algorithm

The outlier detection algorithm used is given below:

```
def remove_outliers(data, pp1, pp2):
"""
Based on "Data Outlier Detection using the Chebyshev Theorem",
Brett G. Amidan, Thomas A. Ferryman, and Scott K. Cooley
Keyword arguments:
data -- A numpy array of discrete or continuous data
pp1 -- likelihood of expected outliers (e.g. 0.1, 0.05 , 0.01)
pp2 -- final likelihood of real outliers (e.g. 0.01, 0.001 , 0.0001)
"""
mu1 = np.mean(data)
sigma1 = np.std(data)
k = 1./ np.sqrt(pp1)
odv1u = mu1 + k * sigma1
odv1l = mu1 - k * sigma1
new_data = data[np.where(data <= odv1u)[0]]
new_data = new_data[np.where(new_data >= odv1l)[0]]
mu2 = np.mean(new_data)
sigma2 = np.std(new_data)
k = 1./ np.sqrt(pp2)
odv2u = mu2 + k * sigma2
odv2l = mu2 - k * sigma2
final_data = new_data[np.where(new_data <= odv2u)[0]]
final_data = new_data[np.where(final_data >= odv2l)[0]]

return final_data
```

The algorithm used for extracting the final positions from the 50 samples for each position is given below. The example given below is used for extracting position data of the large object when placed in the forward by the arm.

```
#large_straight
path = r'large_straight' # use your path
all_files = glob.glob(path + "/*.csv")

li = []

for filename in all_files:
df = pd.read_csv(filename, index_col=None, skiprows=0)
li.append(np.array(df))
li = np.array(li).reshape(20, 50, 4)

mean_list=[]
for value in range(len(li)):
xmean=li[value][:,0].mean()
ymean=li[value][:,1].mean()
tmean=li[value][:,2].mean()
mean_list.append([xmean, ymean, tmean])
large_straight= np.array(mean_list)
```

4.4 Observations

- Among the all the objects provided the longest one took the most time to settle as comparison to the shortest one because the height of center of gravity.
- As we have observed from the readings that uncertainty decreases with the height of the metal bar.
- With the increase in time, the youbot motors gain heat and this may cause some uncertainty.

Given below are the plots of the right, left and straight positions of the large, medium and small-sized object moved by the robot arm **without** removing the outliers.

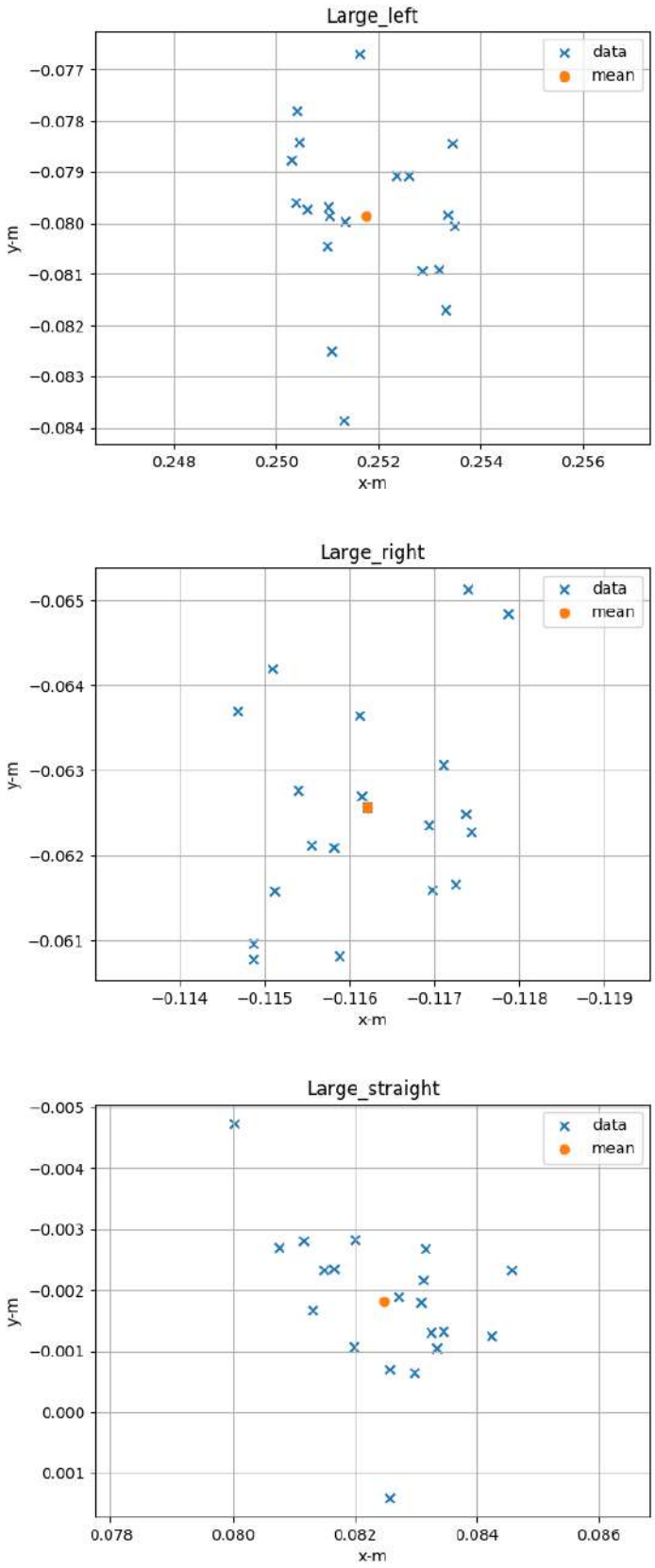


Figure 22: Position Visualization for Large-Sized Object

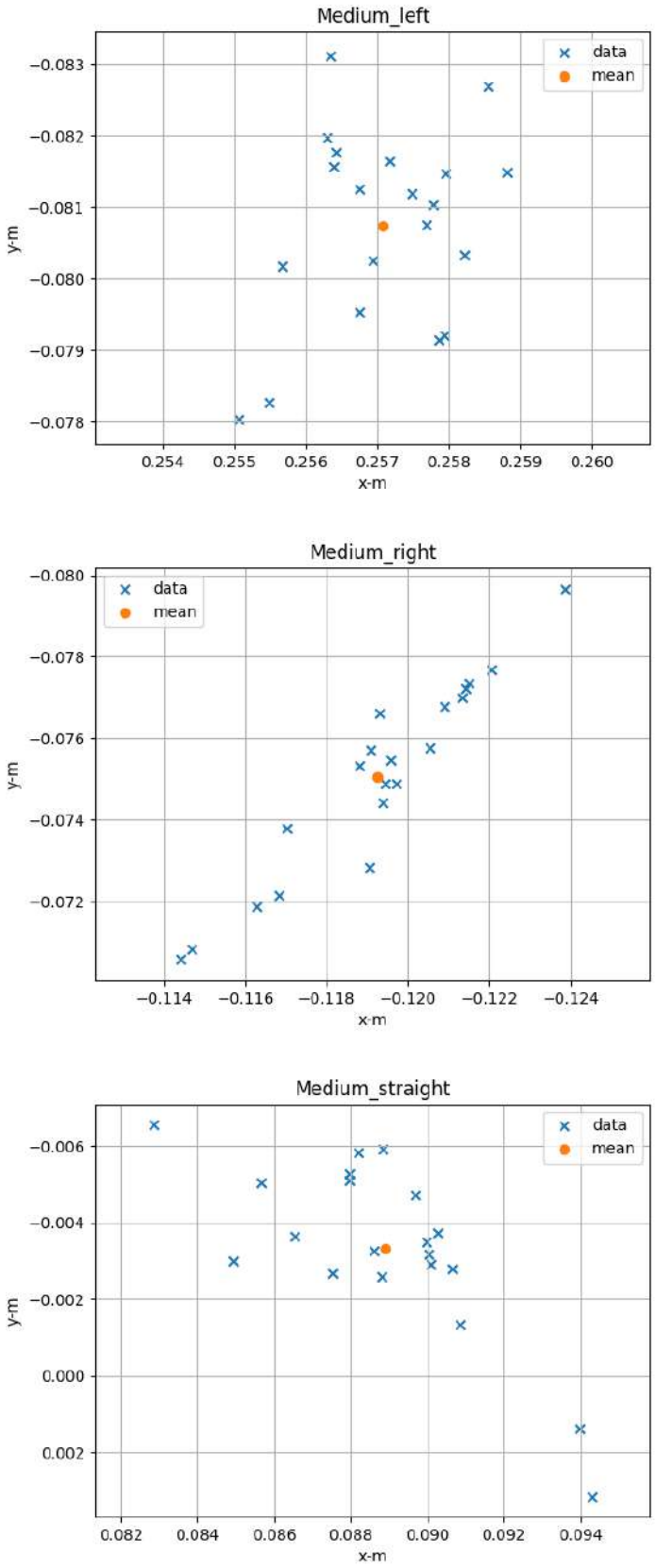


Figure 23: Position Visualization for Medium-Sized Object

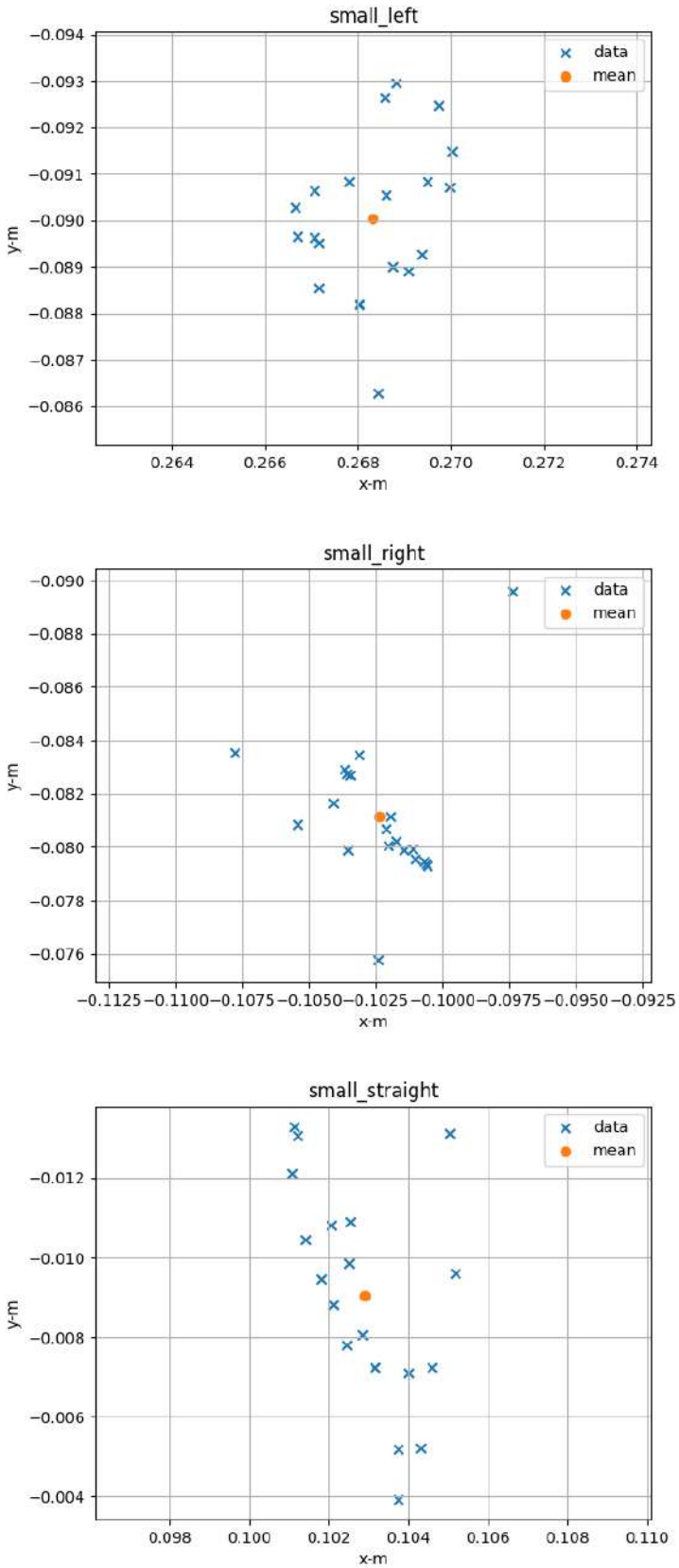


Figure 24: Position Visualization for Small-Sized Object

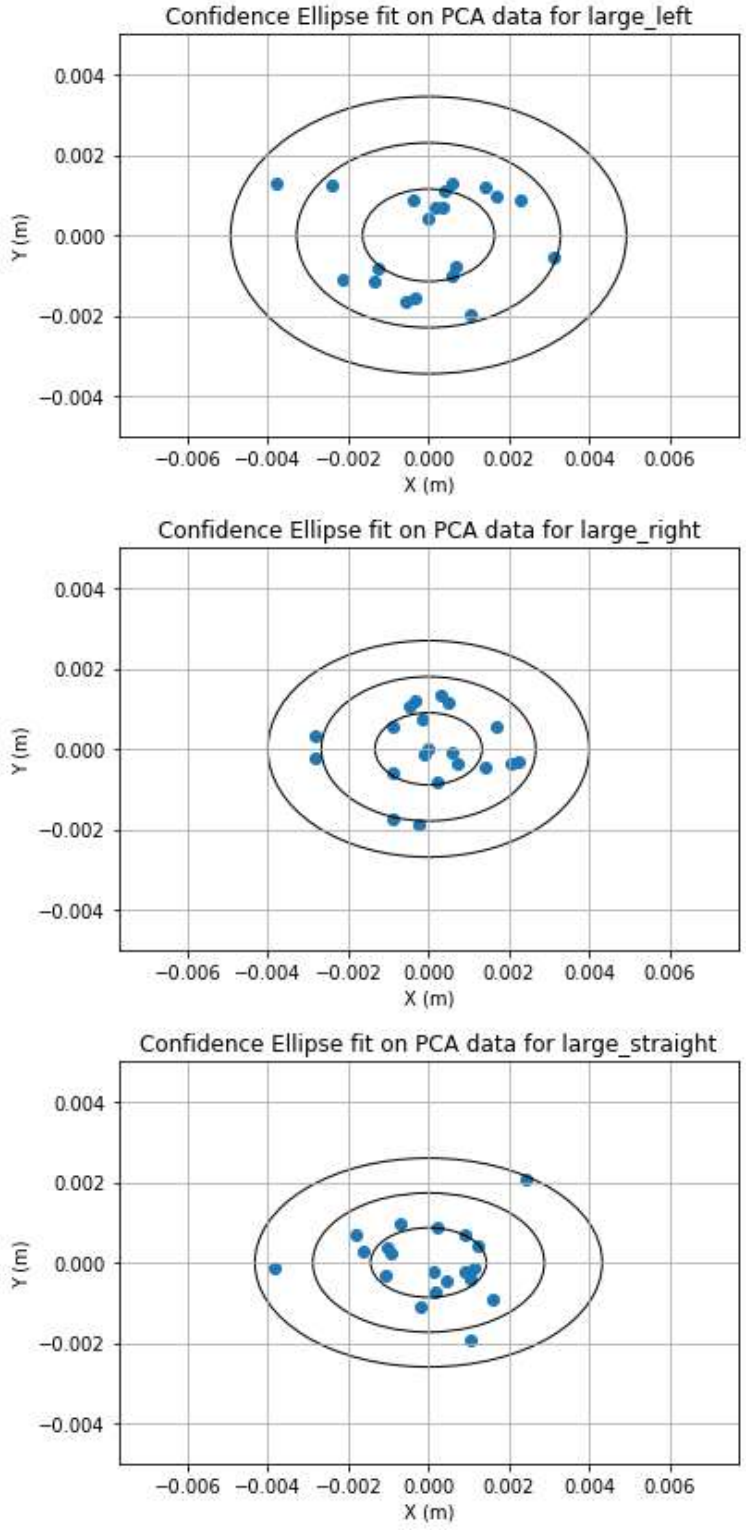


Figure 25: PCA for Left Motion of Large-Sized Object

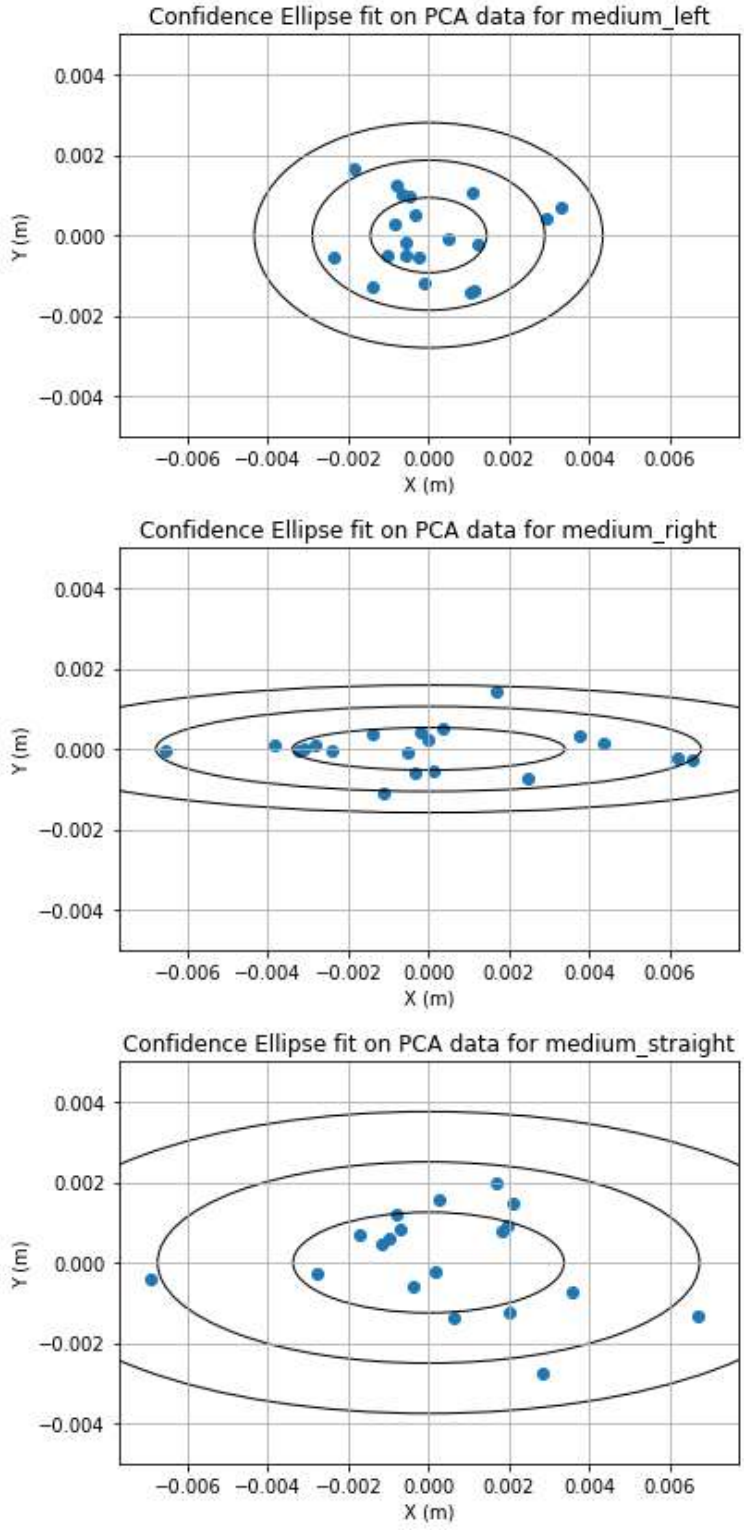


Figure 26: PCA for Right Motion of Large-Sized Object

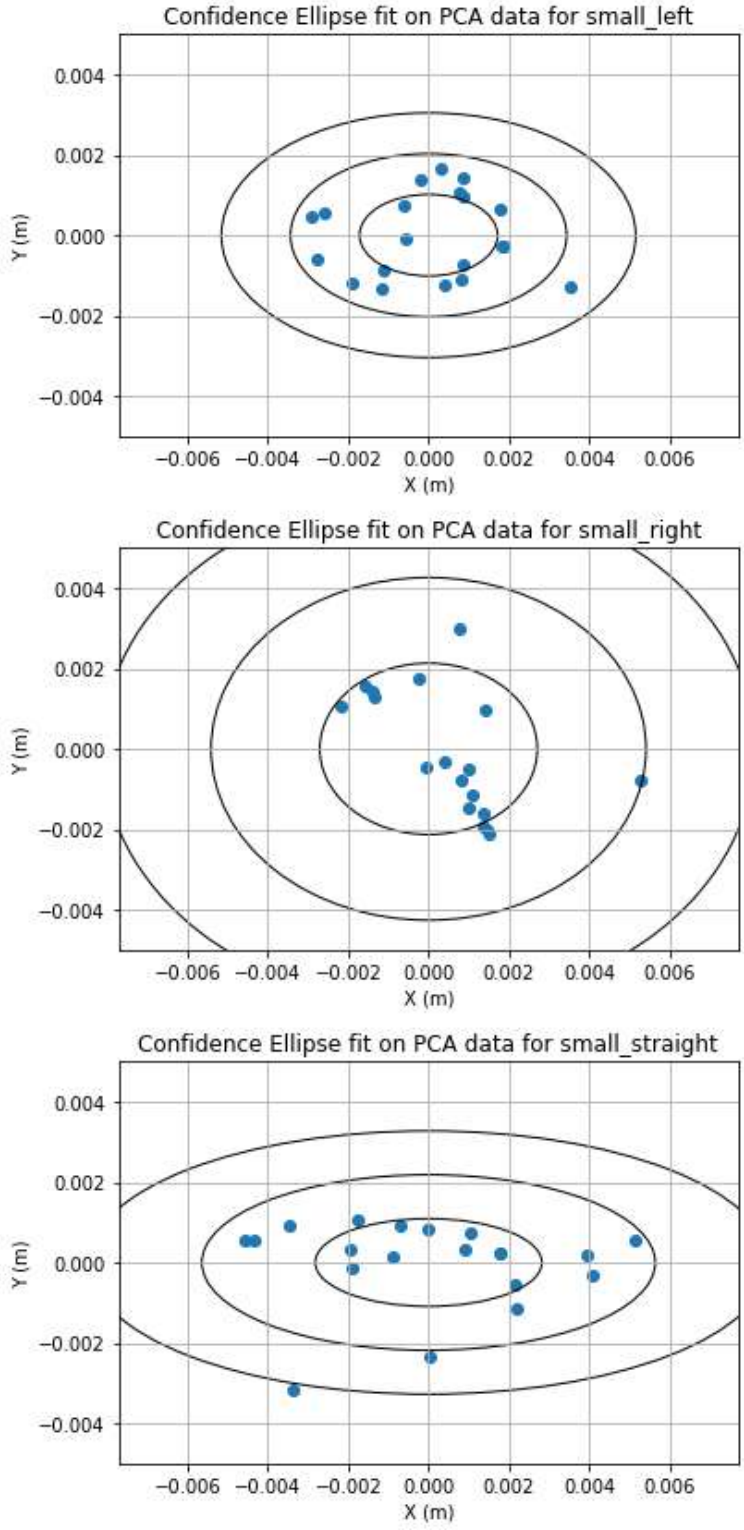


Figure 27: PCA for Straight Motion of Large-Sized Object

Given below are the plots of the right, left and straight positions of the large, medium and small-sized object moved by the robot arm **after** removing the outliers.

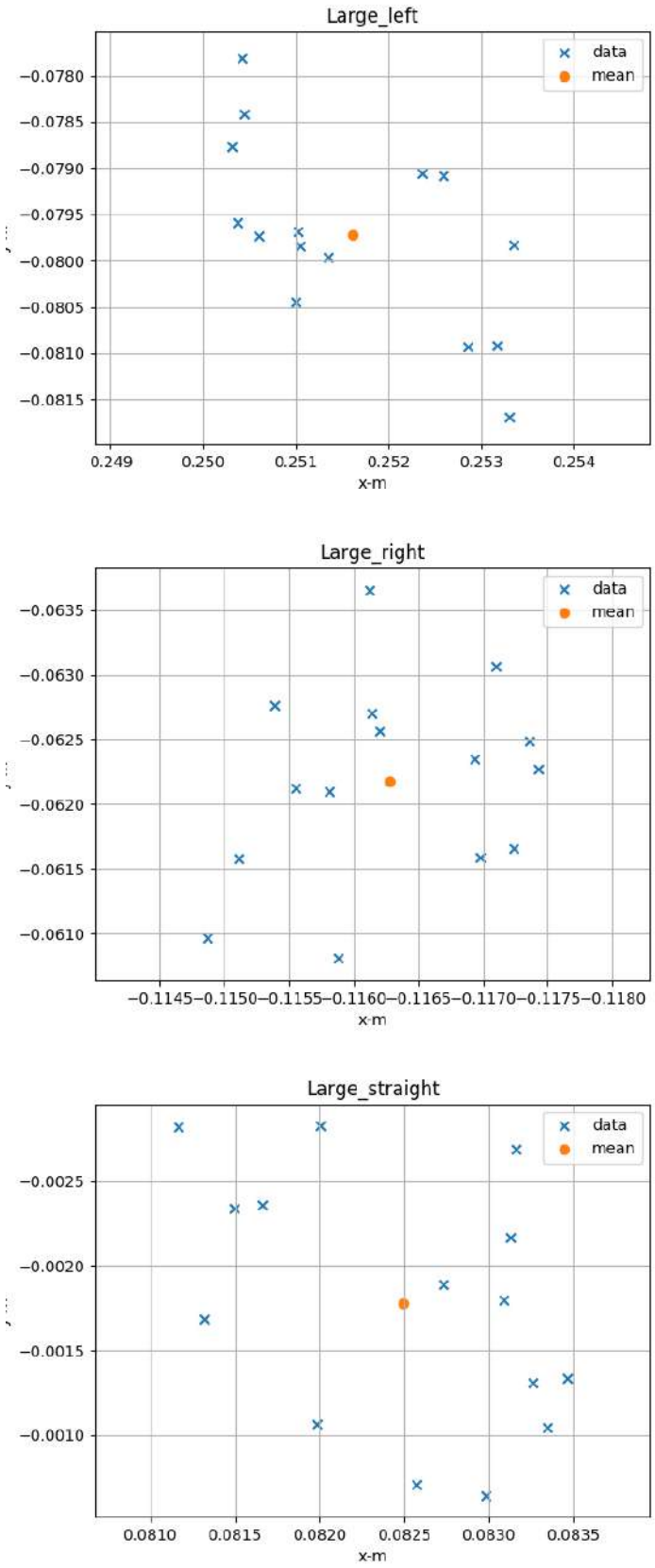


Figure 28: Position Visualization for Large-Sized Object after Outlier Removal

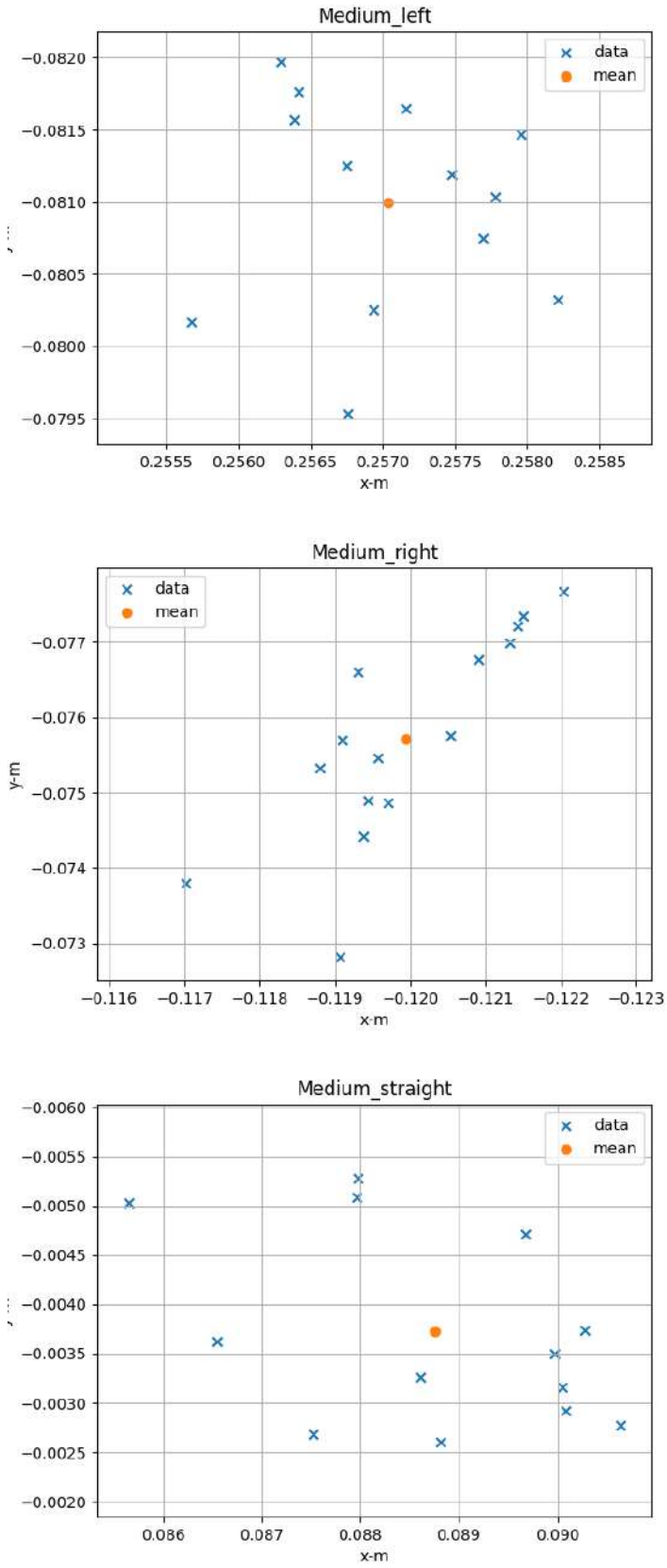


Figure 29: Position Visualization for Medium-Sized Object after Outlier Removal

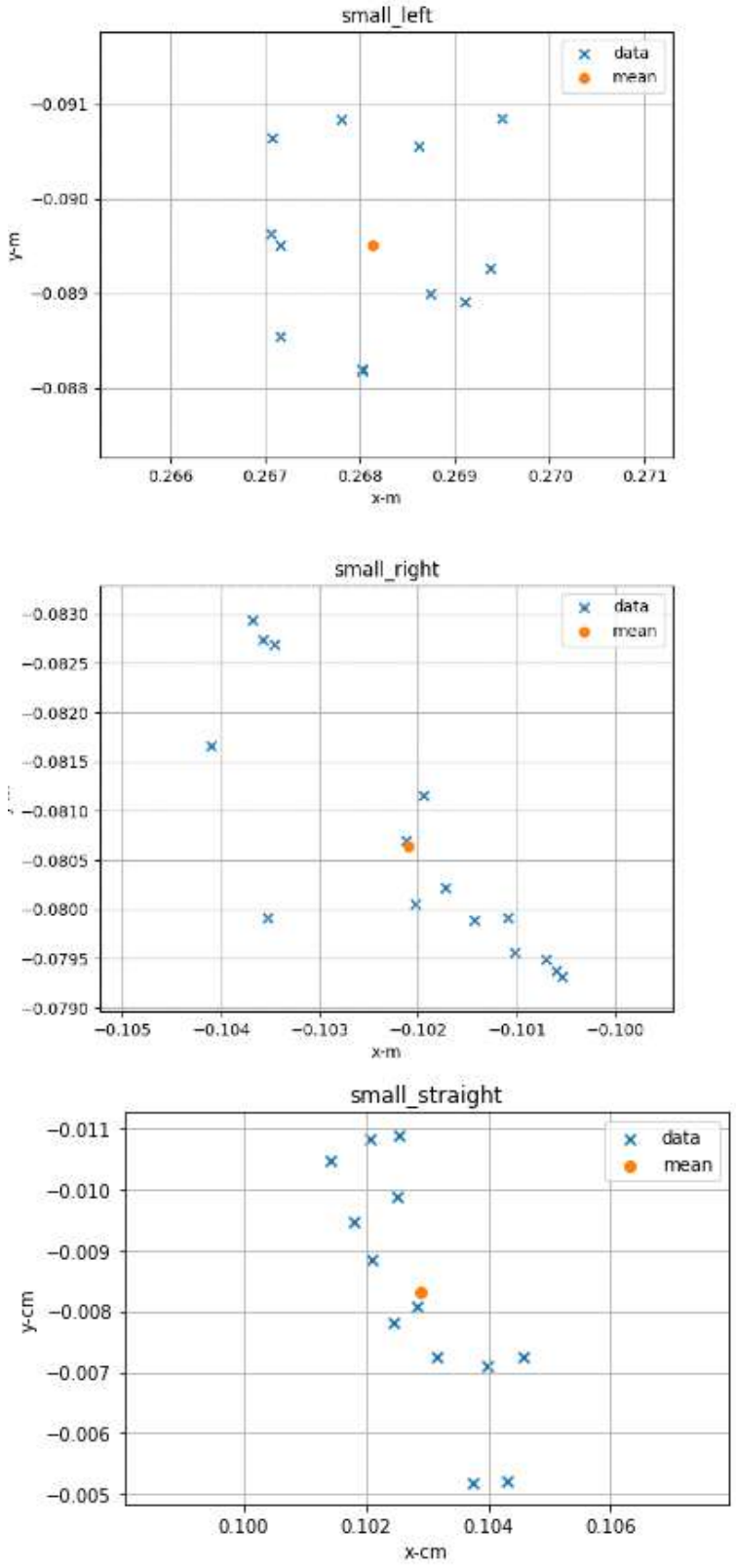


Figure 30: Position Visualization for Small-Sized Object after Outlier Removal

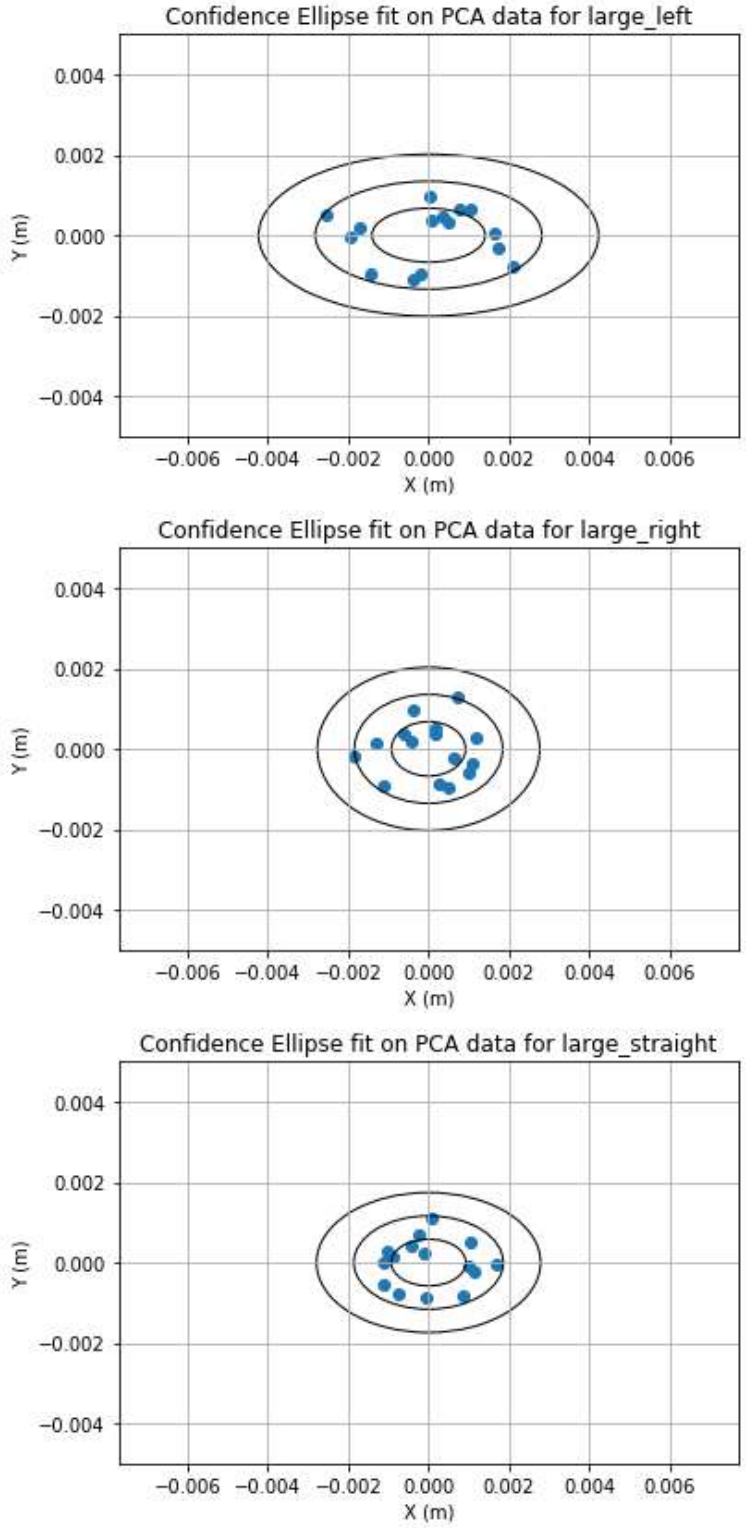


Figure 31: PCA for Large-Sized Object after Outlier Removal

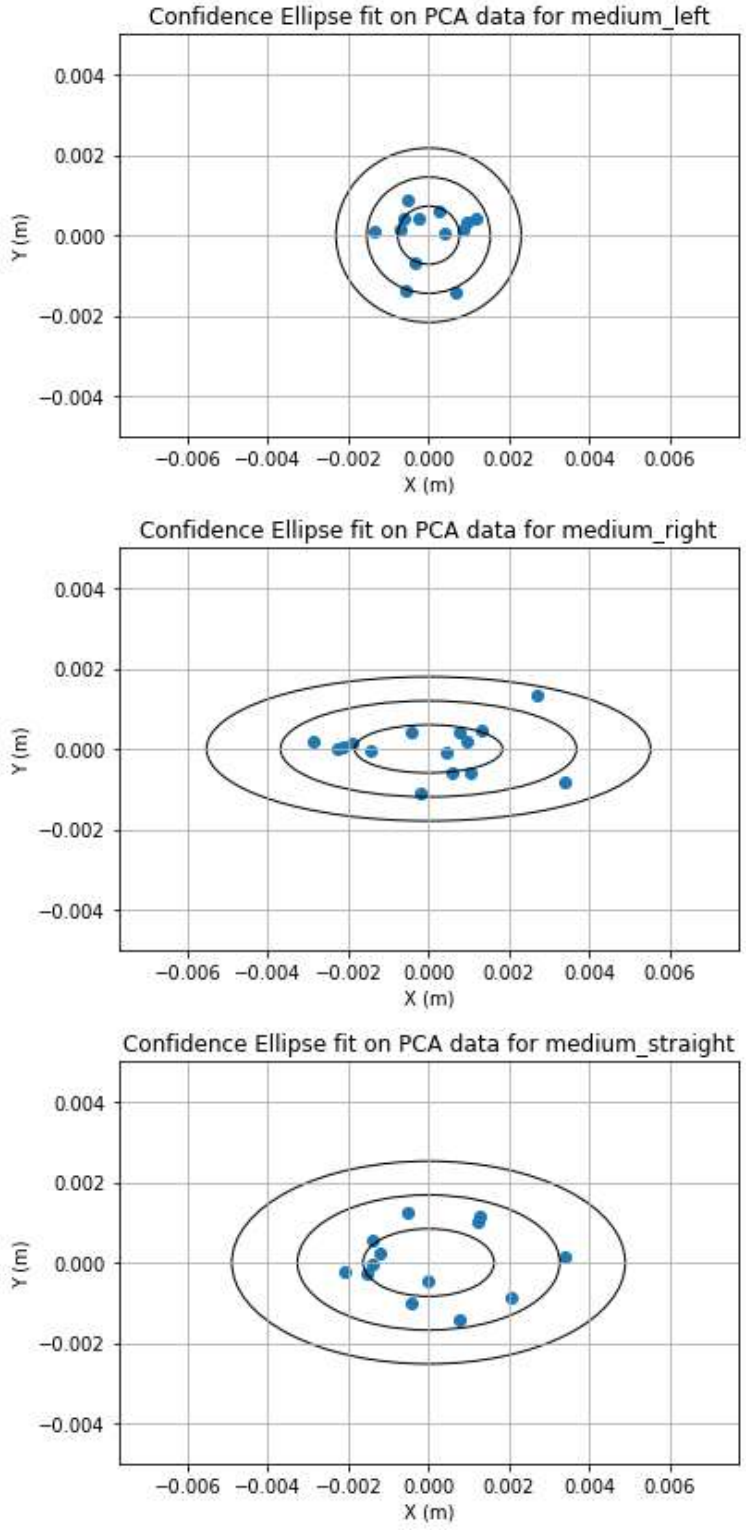


Figure 32: PCA for Medium-Sized Object after Outlier Removal

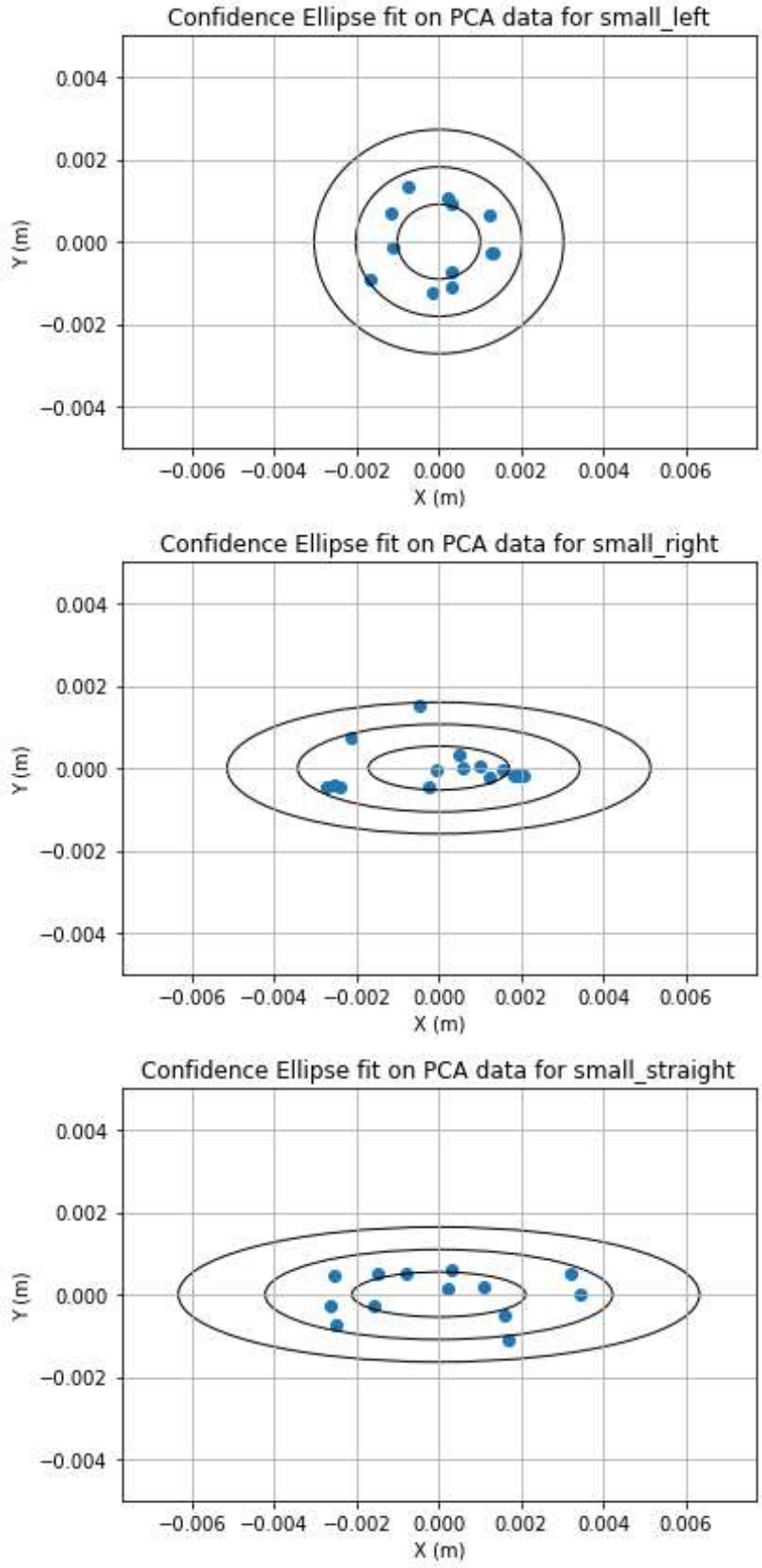


Figure 33: PCA for Small-Sized Object after Outlier Removal

In order to check if the final positions fit the gaussian distribution, the following plots have been made .

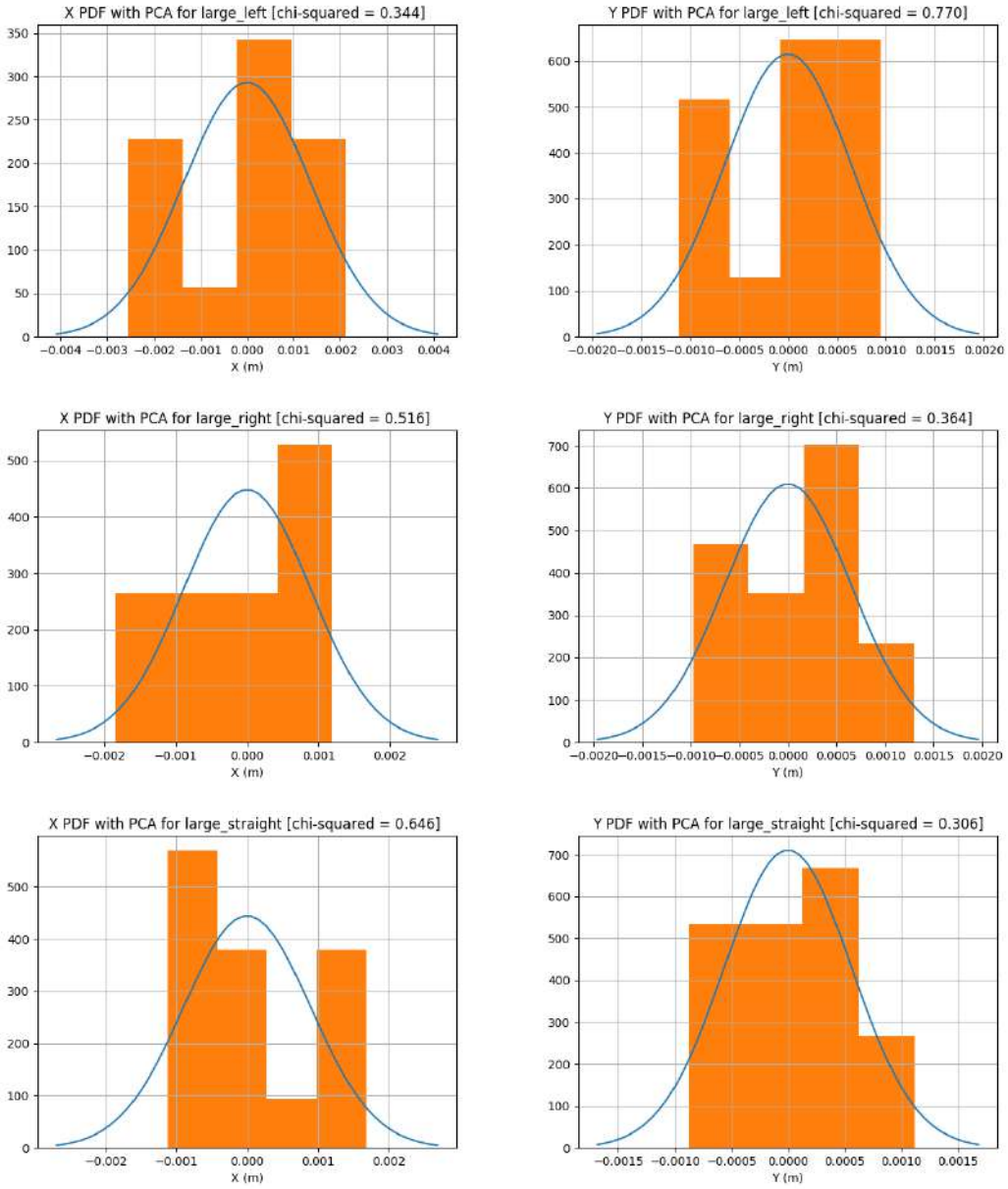


Figure 34: Checking the gaussian fit for Large-sized object

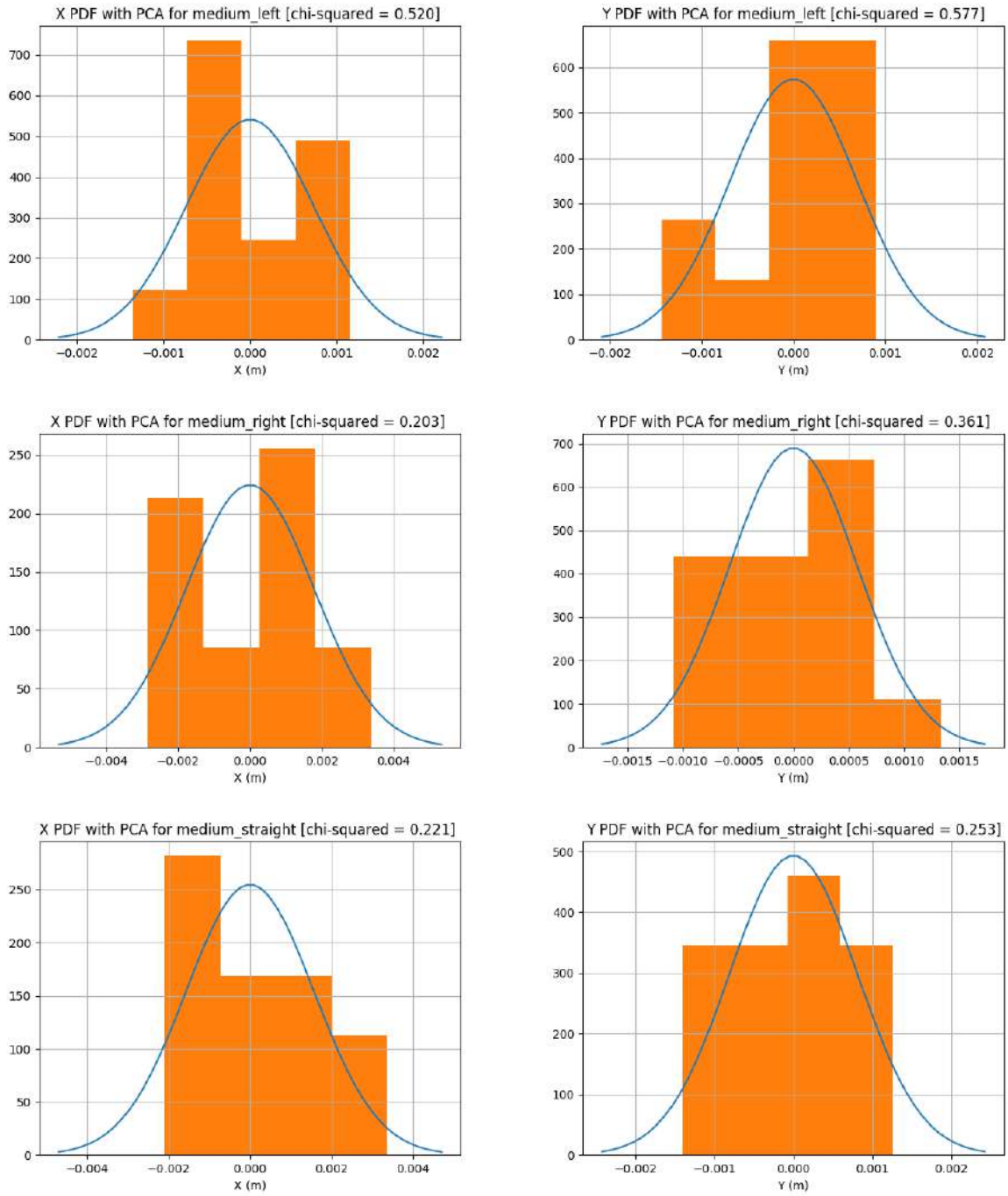


Figure 35: Checking the gaussian fit for Medium-sized object

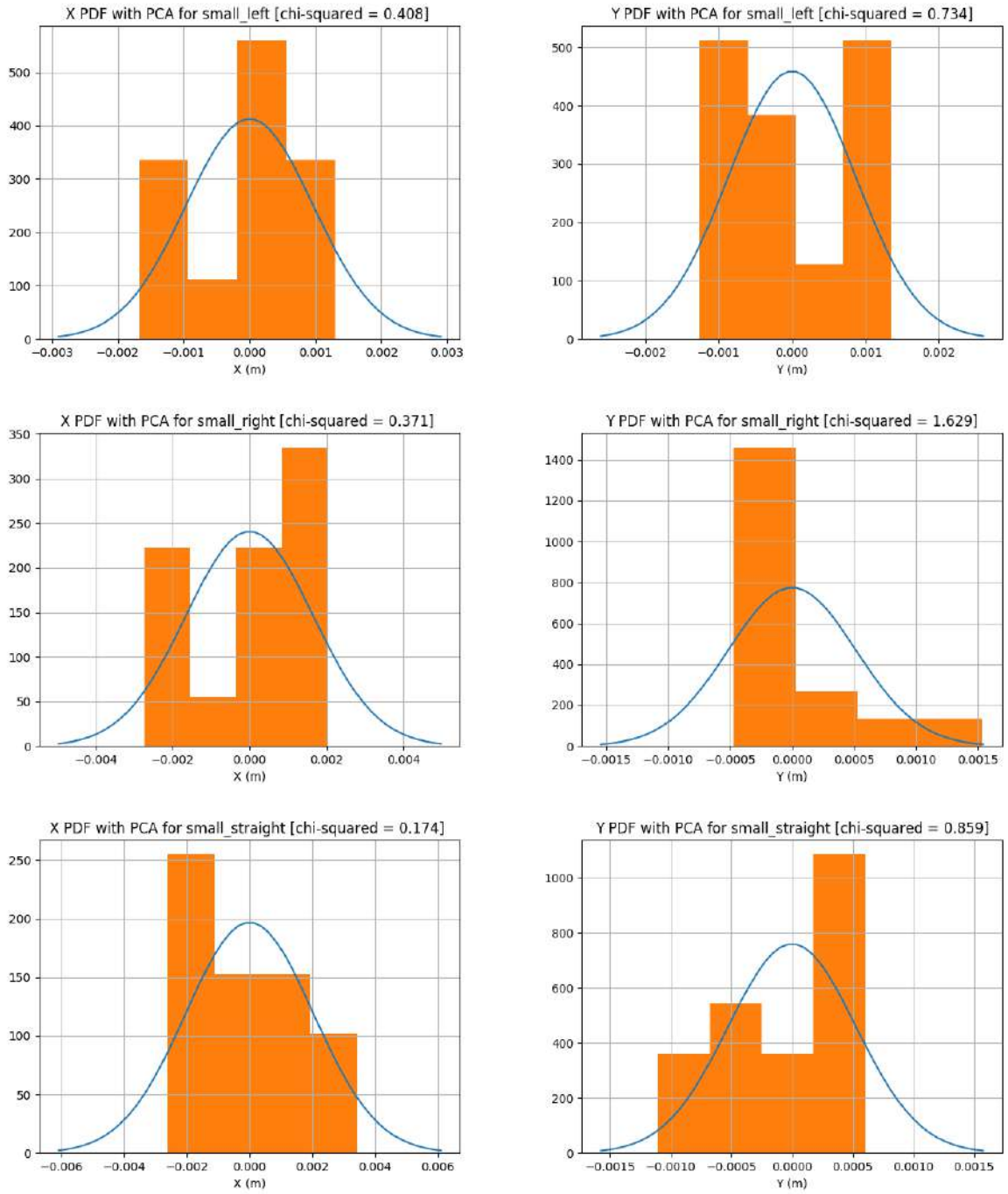
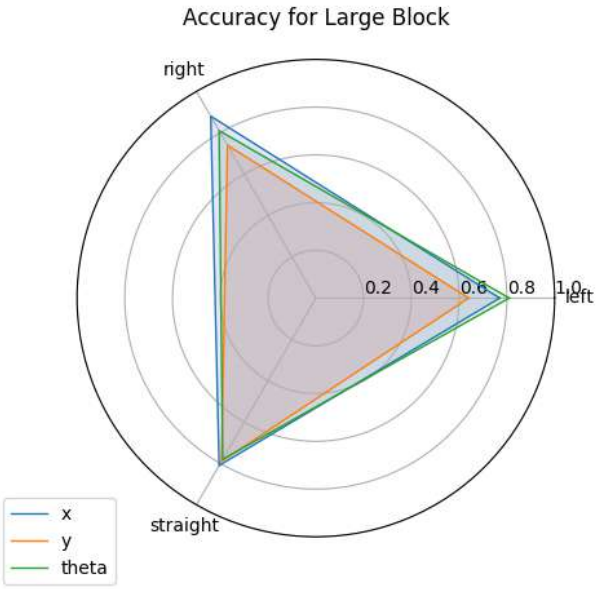


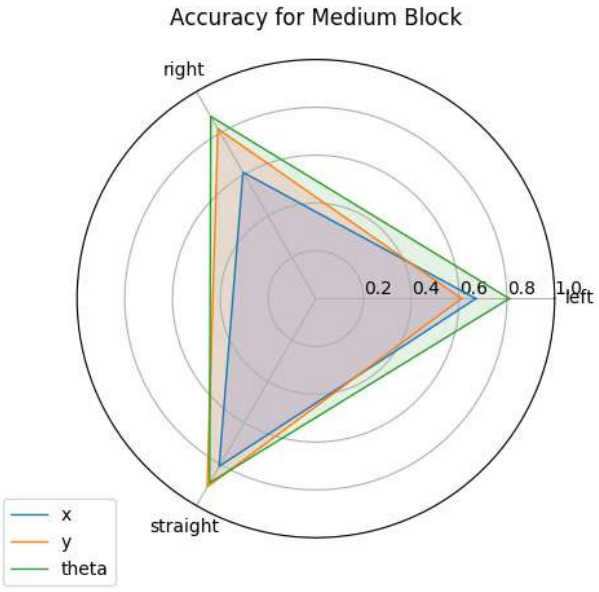
Figure 36: Checking the gaussian fit for Small-sized object

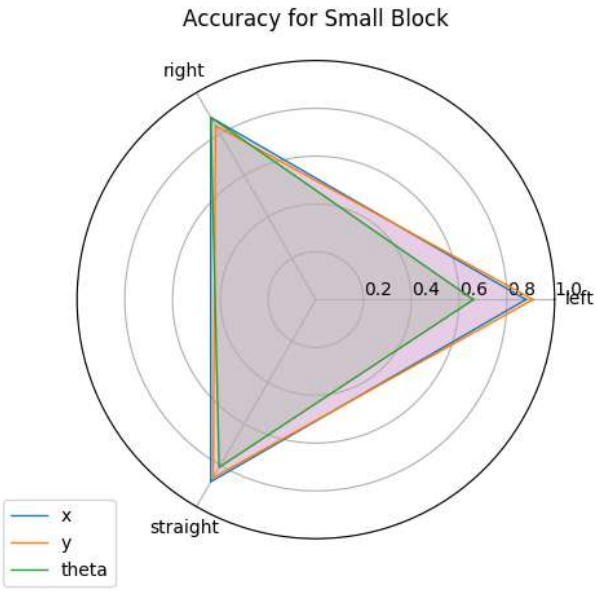
4.4.1 Accuracy and Precision

The accuracy and precision values for the poses are given below.

$$\textbf{Accuracy} = \frac{abs(desired.position - avg.of.actual.position)}{desired.position}$$







Position	X	Y	Orientation
Left	0.77	0.64	0.81
Right	0.88	0.74	0.81
Straight	0.81	0.79	0.78

Table 3: Accuracy Values for large sized object

Position	X	Y	Orientation
Left	0.67	0.61	0.81
Right	0.61	0.82	0.88
Straight	0.81	0.91	0.89

Table 4: Accuracy Values for medium sized object

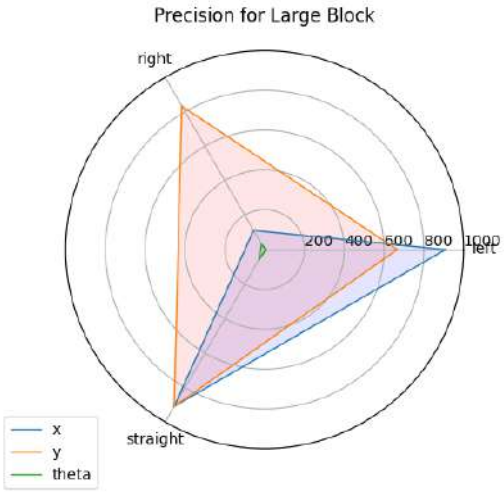
Position	X	Y	Orientation
Left	0.88	0.91	0.66
Right	0.88	0.84	0.88
Straight	0.88	0.86	0.81

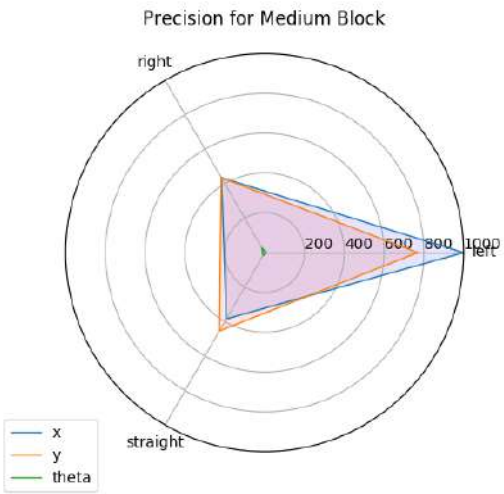
Table 5: Accuracy Values for small sized object

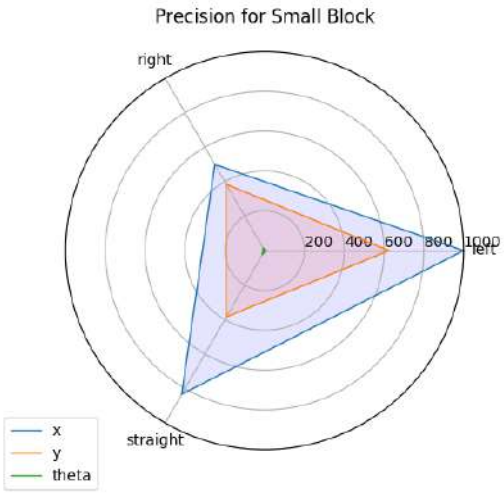
Conclusion:

- From the given formula, lower the calculated value, better is the accuracy of the measurement.
- In the straight motion, the accuracy values are best for large block compared to the other blocks.
- In the right motion, the accuracy for medium block is better in the x-direction compared to the other blocks but it is best for the large block in the y-direction and theta.
- In the left motion, both large and medium blocks have better accuracy in the y-direction, medium in the x-direction and small block for theta.

$$\textbf{Precision} = \frac{1}{\sigma(actual.pos)}$$







Position	X	Y	Orientation
Left	909.09	666.66	7.19
Right	111.11	833.33	29.85
Straight	909.09	909.09	59.17

Table 6: Precision Values for large sized object

Position	X	Y	Orientation
Left	1000	769.23	6.49
Right	434.78	434.78	27.39
Straight	384.61	454.54	17.03

Table 7: Precision Values for medium sized object

Position	X	Y	Orientation
Left	1000	625	6.29
Right	500	384.61	14.66
Straight	833.33	384.61	22.88

Table 8: Precision Values for small sized object

Conclusion:

- From the given formula, higher the calculated value, better is the precision of the measurement.
- In all blocks, theta has very less precision for all motions.
- In the straight motion, the large block has much better precision compared to the others in the y-direction and similar precision to the small block in the x-direction.
- In the right motion, the precision in the y-direction is best for large block and in the x-direction is best for small block.
- In the left motion, both medium and small blocks have best precision in the x-direction and medium block has better precision in the y-direction.

4.4.2 Possible Errors

- Towards the end of the experiment, the robot arm may not function as precisely as it moved in the beginning since the motors etc. might get heated up due to continuous use. This could be a source of error in the position of the object placed.
- When the object is placed at the initial position, there is a tolerance provided within the frame setup, which could cause an error.
- Since the object is dropped from a height, this could cause a slight displacement from the intended position while placing it.

4.5 Hypothesis Testing

To check if the hypothesis holds true, Welch T-test is used to compare the data. We have used two different set of hypothesis:

- H_0 : The null hypothesis for the test is that the means are equal.
- H_A : The alternate hypothesis for the test is that means are not equal.

To check for the hypothesis, we need a p-value and for which we need the information of variances (of the population and the sample) and the mean values of both the population and the sample. The method is to be performed using built-in functions from the scipy library, "stats.ttest".

Straight Motion		
Small and Medium Weights		
	T-test value	p-value
x (m)	20.80945991	0.00E+00
y (m)	-7.07935476	9.98E-01
theta (degrees)	-4.44472951	9.26E-03
Small and Large Weights		
	T-test value	p-value
x (m)	52.49206886	0.99991615
y (m)	-10.79415914	0.01992778
theta (degrees)	1.29615955	0.01385102
Medium and Large Weights		
	T-test value	p-value
x (m)	9.69534546	1.20E-01
y (m)	-2.54836587	1.82E-01
theta (degrees)	6.32178973	1.52E-04

Figure 37: Straight Motion T-Test Results

Right Motion		
Small and Medium Weights		
	T-test value	p-value
x (m)	23.28871992	0.00E+00
y (m)	-7.53390252	5.54E-01
theta (degrees)	-4.15633123	3.98E-01
Small and Large Weights		
	T-test value	p-value
x (m)	26.00155029	0.00E+00
y (m)	-28.01173412	0.00E+00
theta (degrees)	-4.98215105	2.51E-01
Medium and Large Weights		
	T-test value	p-value
x (m)	-5.1936538	2.22E-01
y (m)	-20.4420921	0.00E+00
theta (degrees)	-1.14838279	4.78E-01

Figure 38: Right Motion T-Test Results

Left Motion		
Small and Medium Weights		
	T-test value	p-value
x (m)	33.17794568	0.00E+00
y (m)	-19.28491407	0.00E+00
theta (degrees)	-1.29581698	4.68E-03
Small and Large Weights		
	T-test value	p-value
x (m)	45.89710606	1.00E+00
y (m)	-19.59110704	0
theta (degrees)	0.83755973	3.57E-02
Medium and Large Weights		
	T-test value	p-value
x (m)	15.12312947	0.00E+00
y (m)	-1.85583028	9.95E-01
theta (degrees)	2.23419054	1.17E-02

Figure 39: Left Motion T-Test Results

4.5.1 Results

As it can be seen the ttest function assumes the population to have identical variances by default. Accordingly the test measures if the average (expected) value differs significantly across samples. So we observe if the p-value is no smaller than 0.03 then we reject the null hypothesis H_0 is rejected. As a result form the obtained values as shown above, we conclude that:

- The p-value differs in comparison to different weights.
- A majority of the null hypothesis are rejected.
- The weights of different objects play a significant part in final pose.

Reasons:

Due to momentum gained from different weights, the heavier object have imprecise effects on the final pose than that of the light weight. Also if the distance between center of gravity of the object and the ground is high, the objects are more unstable. Thus the height of the metal bar also plays an important role in deciding uncertainty.

4.5.2 Single vs Multiple Measurements

We perform a simple experiment to prove that if we use a single camera measurement for determining the final object pose instead of multiple filtered measurements, the effect on the final pose estimates is significant. We perform the experiment for the left motion of the large block as given below.

1. From the raw camera measurements for the left motion of the large block, randomly choose a single measurement from the 50 measurements of a single sample.
2. Repeat this for all the 20 samples.
3. Compute the standard deviation of these values.
4. Compare this with the standard deviation of the samples derived from the mean of 50 measurements.

Measurement	**X**	**Y**	**Orientation**
Mean of 50	0.0011	0.0015	0.1390
Single	0.0011	0.0033	0.2365

Table 9: Standard Deviation values for Large-Left motion

From the above table, it can be concluded that the standard deviation increases when only a single measurement is taken compared to when all the 50 measurements are used for the experiment. This causes a significant shift in the final pose estimate of the blocks.