Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

Scientific Experimentation and Evaluation

Lab Reports

Tahir Mehmood

Mihir Mulye

Carlo Wiesse

Course instructor & Teaching assistant

Prof. Dr. Paul G. Plöger

Djordje Vukcevic

June 26, 2019

Contents

1

Manual Motion Observations

1.1 Homework description

1.1.1 Problem: 1

1. On the last three lecture slides, you will find the description of a soft tissue measurement experiment.

2. Apply the appropriate terms from the slide "2B: Formalization general terms" to this description.

1.1.2 Problem: 2

1. Complete assignment 1.1 from the manual and upload deliverable 1.1.

2. Again use the appropriate terms to describe your setup.

1.2 Solution

1.2.1 Problem: 1

1. Measurand
   Displace the material using the indentation tool.

Measure the applied force using FT sensor.
Consider the Simulab complex tissue model and read its response.

2. Measurement
   Axial force was measured.
   Indention depth was measured.

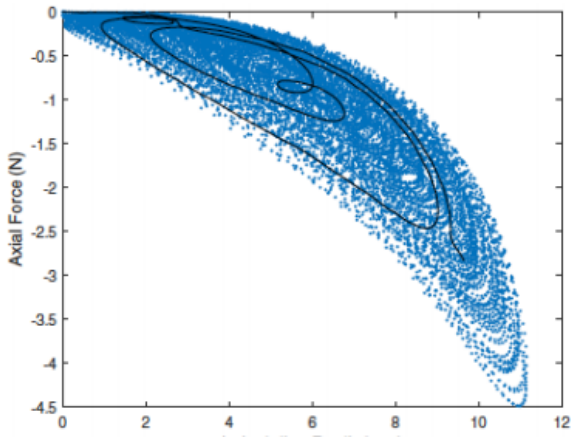
3. Display
   Not provided.

4. (Display) Range
   Not provided.

5. Measurement range
   Not provided.

6. Suppression area:
   The indentation depth was decided to be chosen between 0 - 11.5 mm.

7. Measured (Quantity) Value
   Both axes in a plot. For example, in the following plot, the "measured force" variable is getting values assigned to it on the y-axis. Similarly, the "time" variable is getting values assigned to it on the x-axis.



8. Measurement result: For example: Maximum peak to peak value over this test is 0.0987 N. Standard Deviation of the noise is 0.0042 N.

9. Device Under Test (DUT)
   Simulab complex tissue model.

10. Measurement facility
    The measurement facility is composed by:

    - PI hexapod
    - Monocarrier drive
    - Indentation tool
    - Simulab complex tissue model
    - ATI Mini40 force sensor
    - EPOS2 motor controller

11. Measurement System: The measurement system is composed by:

    - EPOS2 motor controller
    - Indentation tool
    - FT sensor
    - NI DAQ

12. Meter: The meter is composed by:

    - FT sensor
    - NI DAQ

13. Measuring Principle
    Comparison between voltages drop depending on the deformation of silicon strain gauge

14. Measuring method

    (a) Displace the material using the indentation tool.

    (b) Measure the applied force using FT sensor.

    (c) Consider the Simulab complex tissue model and read its response.

15. Sensitivity: Not provided.

1.2.2 Problem: 2

LEGO NXT Experiment Report

1. Aim
   To construct a LEGO Differential Drive Robot and measure the observable end pose variation for three different paths followed by the robot namely arc to left, arc to right and go straight. Repeat the experiment 20 times.

2. Design of the Robot
   The robot is a differential drive assembly with a castor wheel mounted at the back. Two deferentially driven wheels are mounted on the front axle.
   Specifications of the Robot
   Wheel Radius : 2.75 cm
   Go_Straight:
   Number of rotations of wheel = 3.5
   Distance travelled = $2 \times \pi \times r \times 3.5 = 2 \times 3.14 \times 2.75 \times 3.5 = 60.4$ cm
   Measurements are made with respect to the base line (last printed line)

3. Approaches
   Standard Approach

   (a) Place the robot in a specified initial position and then check the position (using marker).

   (b) Run the program and verify whether the robot exhibits the desired behavior.

   (c) Let the execution complete and measure the final position using conventional methods (ruler/measuring tape).

   (d) Record results and calculate errors.

   Aruco Approach

   (a) Place the robot in a specified initial position and then check the position using the camera (using 'Aruco' library).

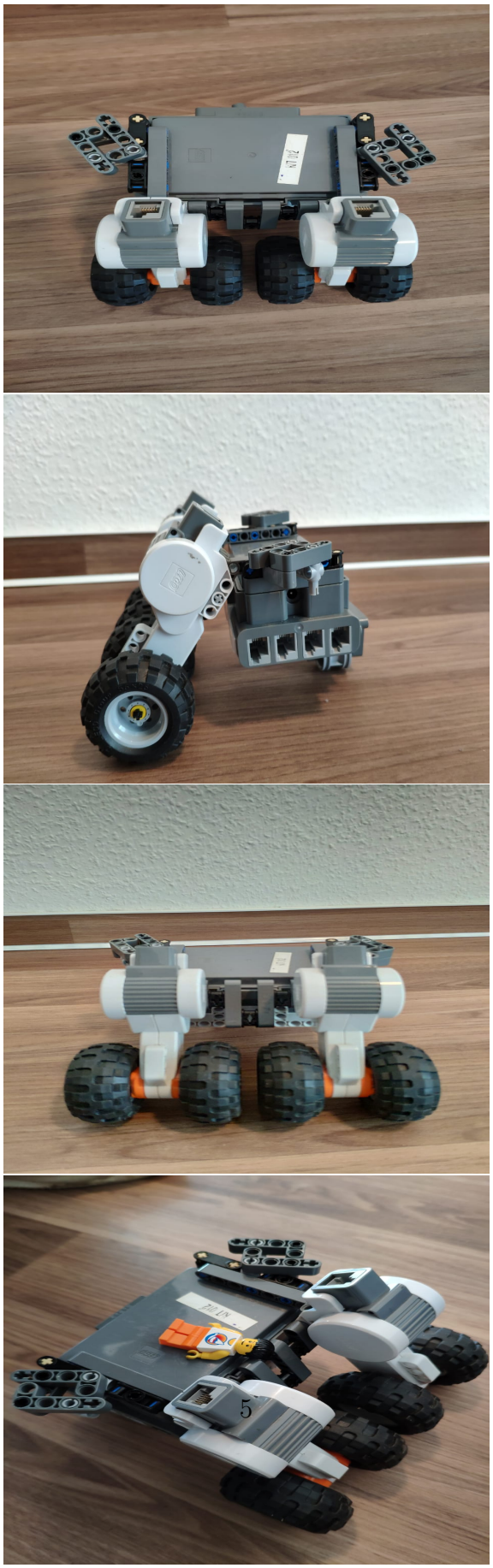


Figure 1.1: Different views of robot view

(b) Run the program and verify whether the robot exhibits the desired behaviour.

(c) Let the execution complete and measure the final position using camera (using 'Aruco' library).

(d) Record results and calculate errors.

4. Problem Description

Problem faced

(a) Replicating initial position over the iterations might be tedious.

(b) Measuring the final pose of the robot might be difficult to accomplish.

(c) The robot might overshoot the final position and the camera might not be able to locate the QR pattern.

Sources of Error

(a) 'Slippage' may take place between the wheels and surface and introduce some error in the final desired location of the robot.

(b) Errors in actuation of motors while turning.

(c) Castor wheel should be aligned properly while initializing the run of the robot to avoid error propagation.

(d) (Case Specific Error) The release of the robot can induce error.
If after starting the program, the robot is not released properly, it can accumulate a lot of error.
Orientation of the robot should be done properly, if not that may be lead to errors.

Possible Solutions

(a) Marking the robot position by a box, so that for the subsequent iterations we are sure of where the robot started.

(b) Making use of Quick Response markers with 'Aruco' package (OpenCV) to eliminate the errors introduced due to variations in initial position.

(c) Using a pen/ruler to mark the initial position of the robot.

5. Experimental Procedure
   For the experiments, the device under test or DUT will be our robot. As for the software, the LEGO Education software was used to develop the programs to test our robot.

   As for the approach used to take measurements, our team decided to go with the "Standard Approach", as described in Section 3.
   Our measurand follows the following steps:

   Step 1: Place the inertial (reference) frame 2.5 cm to the right and 1.2 cm above the bottom left corner of your A0 paper.

   Step 2: Position the robot in the initial pose by doing the following. First, orient the robot in the y direction w.r.t. our inertial frame, then locate the robot at the center of the side of the paper that goes along the x direction w.r.t. our inertial frame, and finally, without dropping any of the last two conditions, place the robot where the Castor wheel lands on top of the x axis of our inertial frame.

   Step 3: Build an auxiliary structure to help you remember the initial pose, as the one shown below.

   Step 4: Additionally, for the same purpose, record the two marker points, which are obtained by you dropping the marker through the right and left marker holders. These two marker points will help us relocate the robot at the beginning of each test run. It is also worth mentioning that these two marker holders are part of the robot and may be a little loose. It is recommended to push the right holder to the back and the left holder to the front (right and left w.r.t. the inertial frame) until these become stable.

   Step 5: Before we start the test runs, you should know how to compute the robot frame. For this, we need to make a few calculations and take some measurements. First, we measure the position of the two marker points w.r.t. our inertial frame, which are composed by an x and y component. Then, we find the middle point of the bounded line described by the two point markers. And finally, we translate this location 3 cm in the y orientation

w.r.t. the robot frame. As for finding our current orientation, we have to rotate the current slope angle by the initial slope angle, which comes from the robot initial pose, along the -z axis of the inertial frame. So, in the end, we are locating our robot frame between our wheels, equidistant from the right and left wheel, laying on the wheel axle line

What we have:

Marker Point 1 (right) $\rightarrow (X_1, Y_1)$

Marker Point 2 (left) $\rightarrow (X_2, Y_2)$
Translation of middle point w.r.t. robot frame $\rightarrow$ [0,3] (cm)
Marker Point 1 initial pose (right) $\rightarrow (x_1, y_1)$
Marker Point 2 initial pose (left) $\rightarrow (x_2, y_2)$
Initial slope angle $\rightarrow \alpha_0 = atan2(Y_1 - Y_2, X_1 - X_2)$
Calculations:

$$P = \begin{bmatrix} (x_1 + x_2)/2 \\ (y_1 + y_2)/2 \end{bmatrix}$$

$$R = \begin{bmatrix} cos(\alpha - \alpha_0) & -sin(\alpha - \alpha_0) \\ sin(\alpha - \alpha_0) & cos(\alpha - \alpha_0) \end{bmatrix}$$

$$\alpha = atan2(y_1 - y_2, x_1 - x_2)$$

$$^{R}_{W}T = \begin{bmatrix} R^T & P + R \begin{bmatrix} 0 \\ 3 \end{bmatrix} \\ 0 & 1 \end{bmatrix}$$

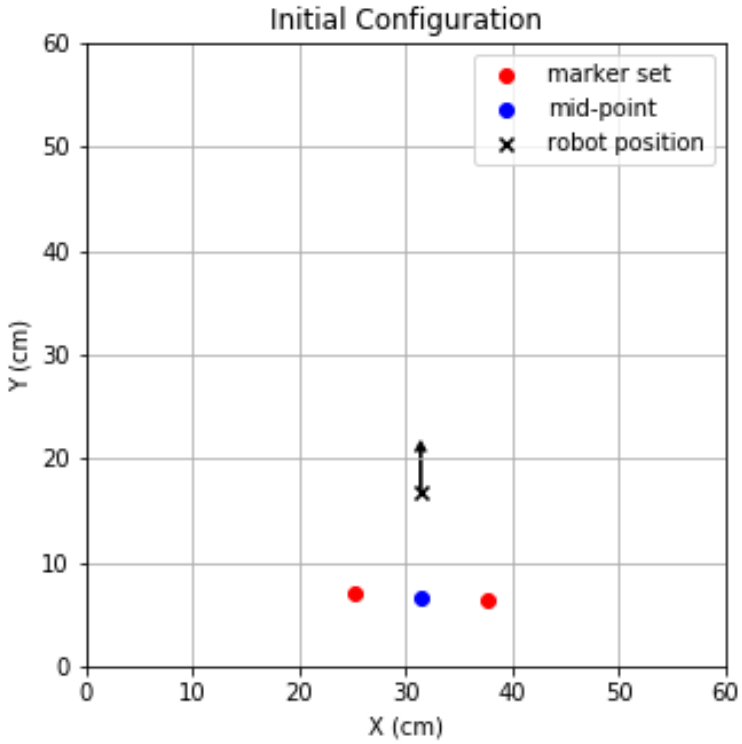


Figure 1.2: Finding the initial robot pose

Step 6: Now that we know how to get the robot frame from the two marker points, we can start running the tests. And for any given 2 set of marker points, we can calculate the current robot pose w.r.t. our inertial frame. Remember to use the auxiliary structure and your recorded marker points at the initial pose to relocate your robot back to the starting pose after every test run.

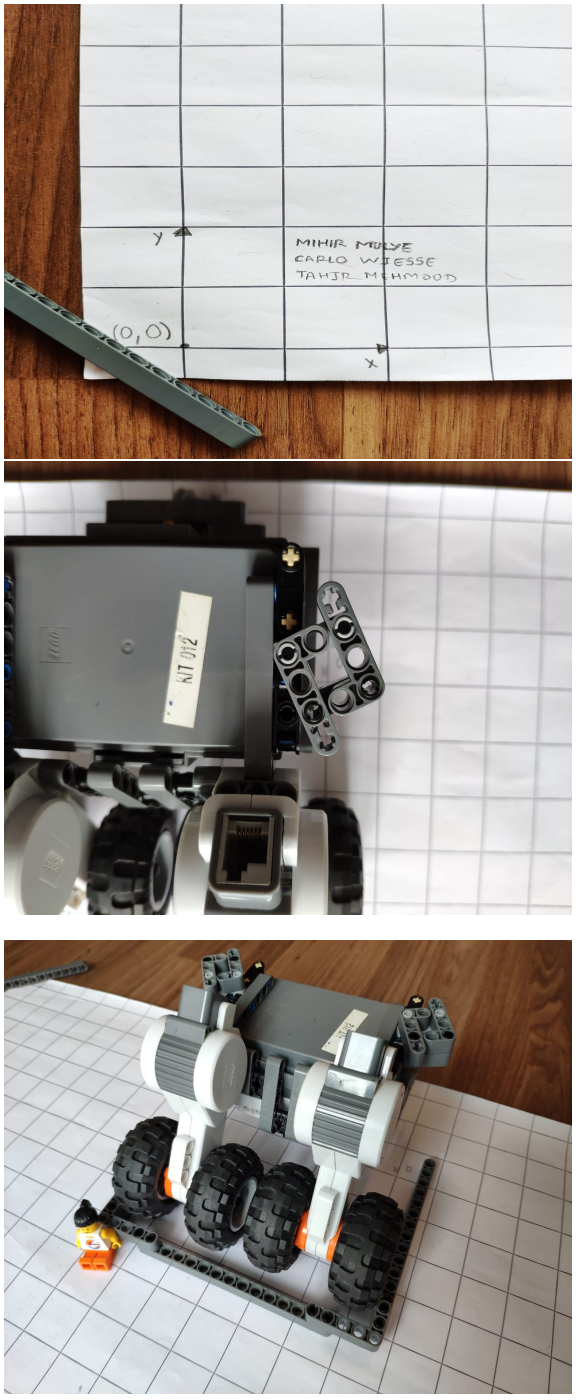


Figure 1.3: Step two for the procedure

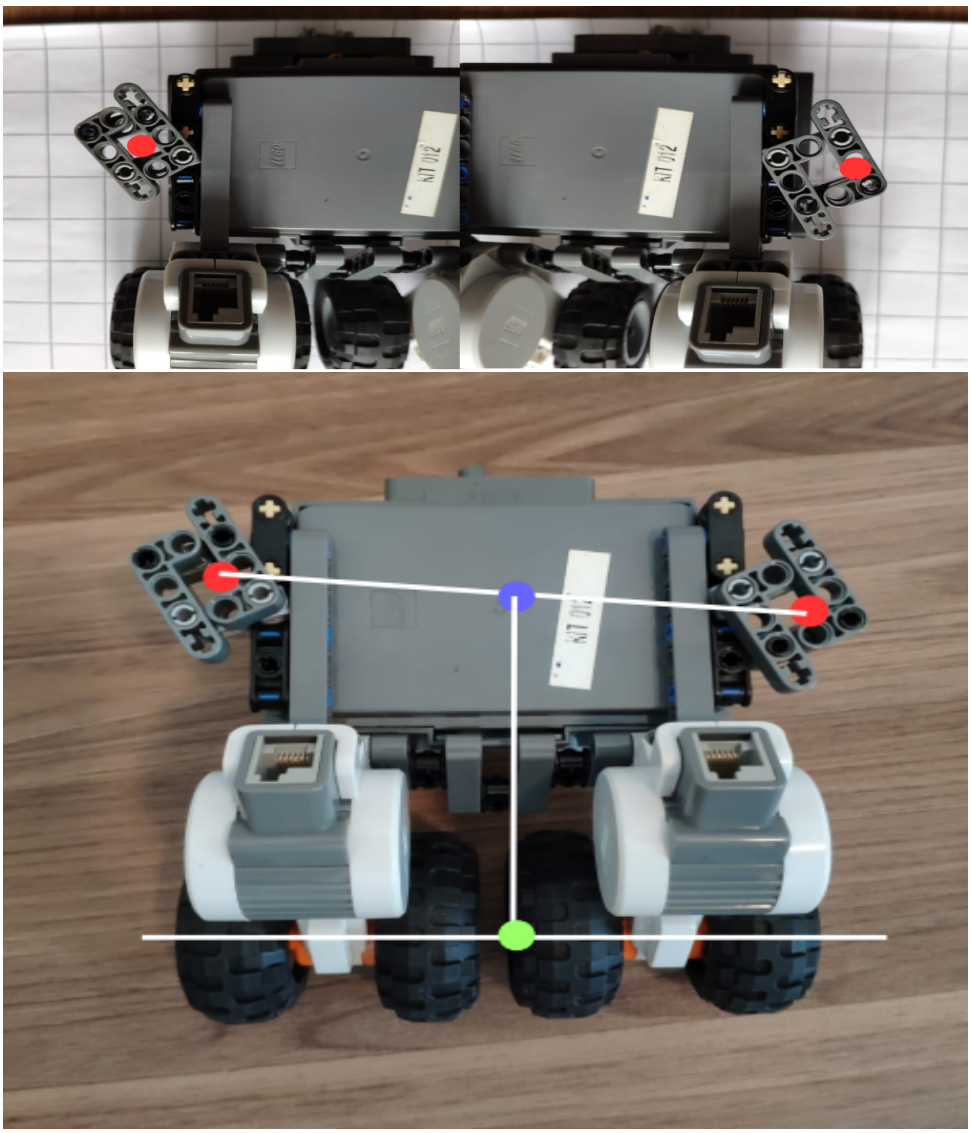


Figure 1.4: Step two for the procedure

6. Experimental Results

- We used three different test and for each 20 values were recorded:
  (a) Straight drive

(b) Left turn

(c) Right turn

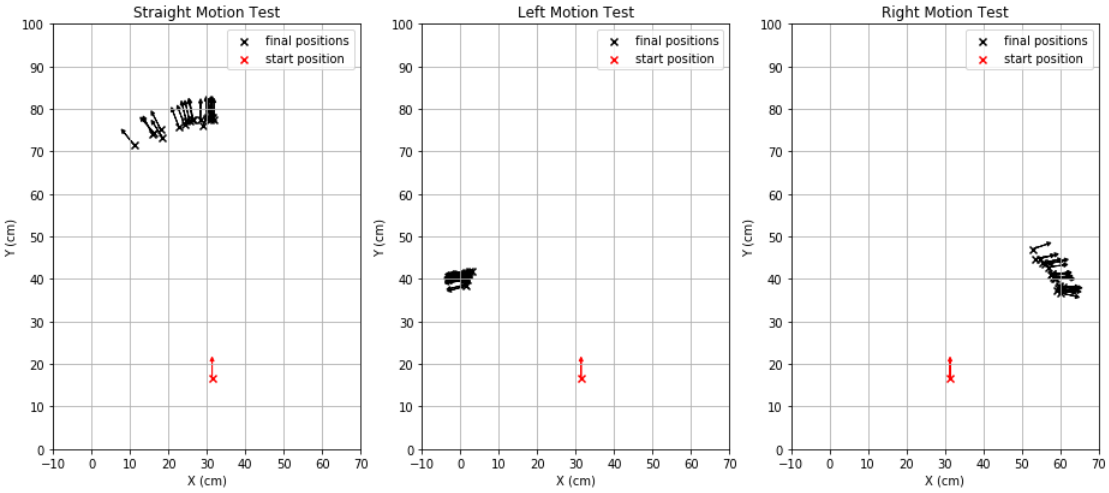


Figure 1.5: Different testing scenarios

7. Chi-squared Test

- The resemblance between our experimental data and the expected Gaussian distribution can be observed in the next figure.
- After testing different values for the number of bins, the number of bins was set to 4.
- As expected, all the chi-squared values are less than 7.5, which gives us a 95% confidence for this fit.

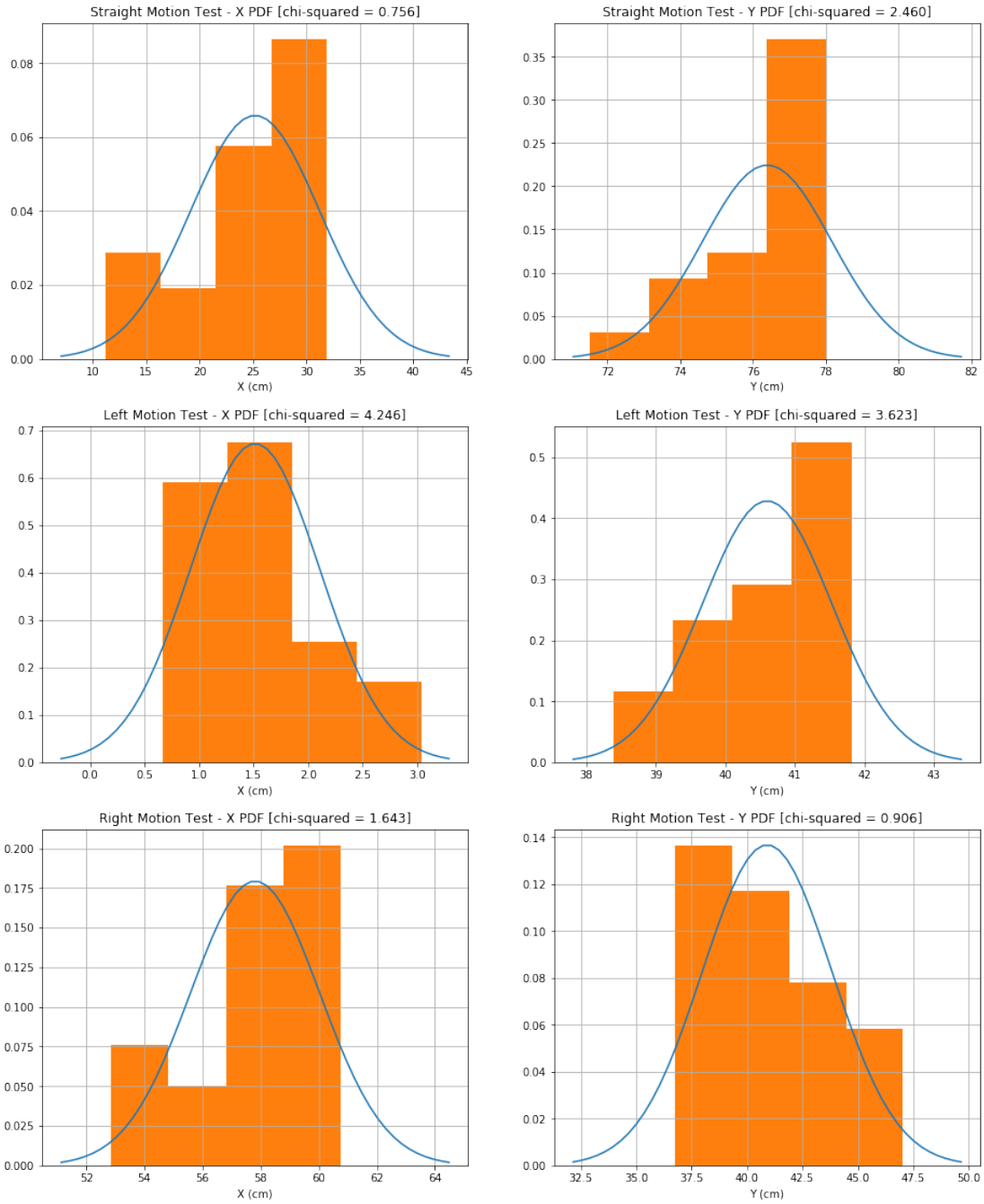


Figure 1.6: Chi-squared Test without PCA

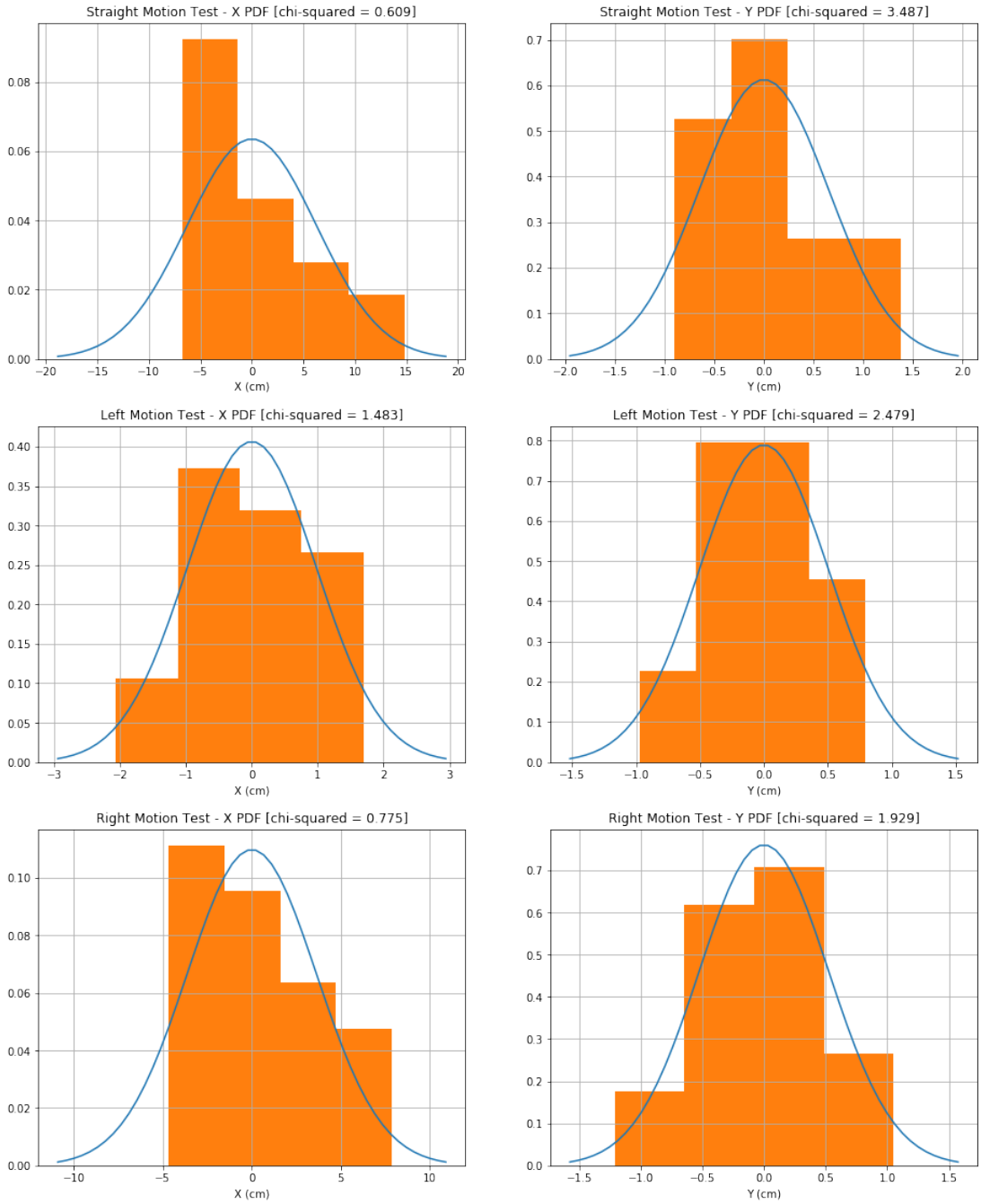


Figure 1.7: Chi-squared Test with PCA

8. Outlier detection

- The PCA data was enclosed in an ellipse defined by the values of stan-

dard deviation in x and y.

- A parameter variable was created to define the outlier limit by increasing the area covered by the ellipse in a proportional manner.
- After testing various values, the selected value for the parameter was set to 2.6. (1.0 being the original ellipse with major axis equal to $2\sigma_x$ and minor axis equal to $2\sigma_y$)

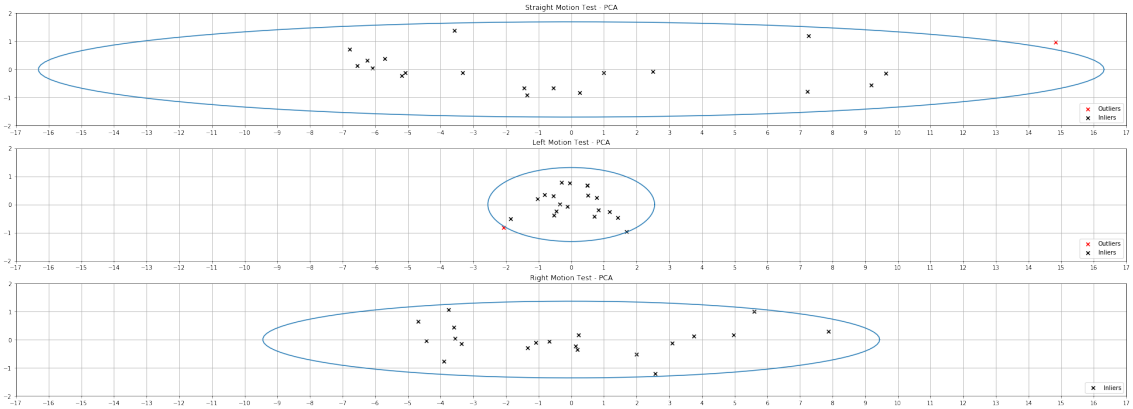


Figure 1.8: Outlier Detection Test

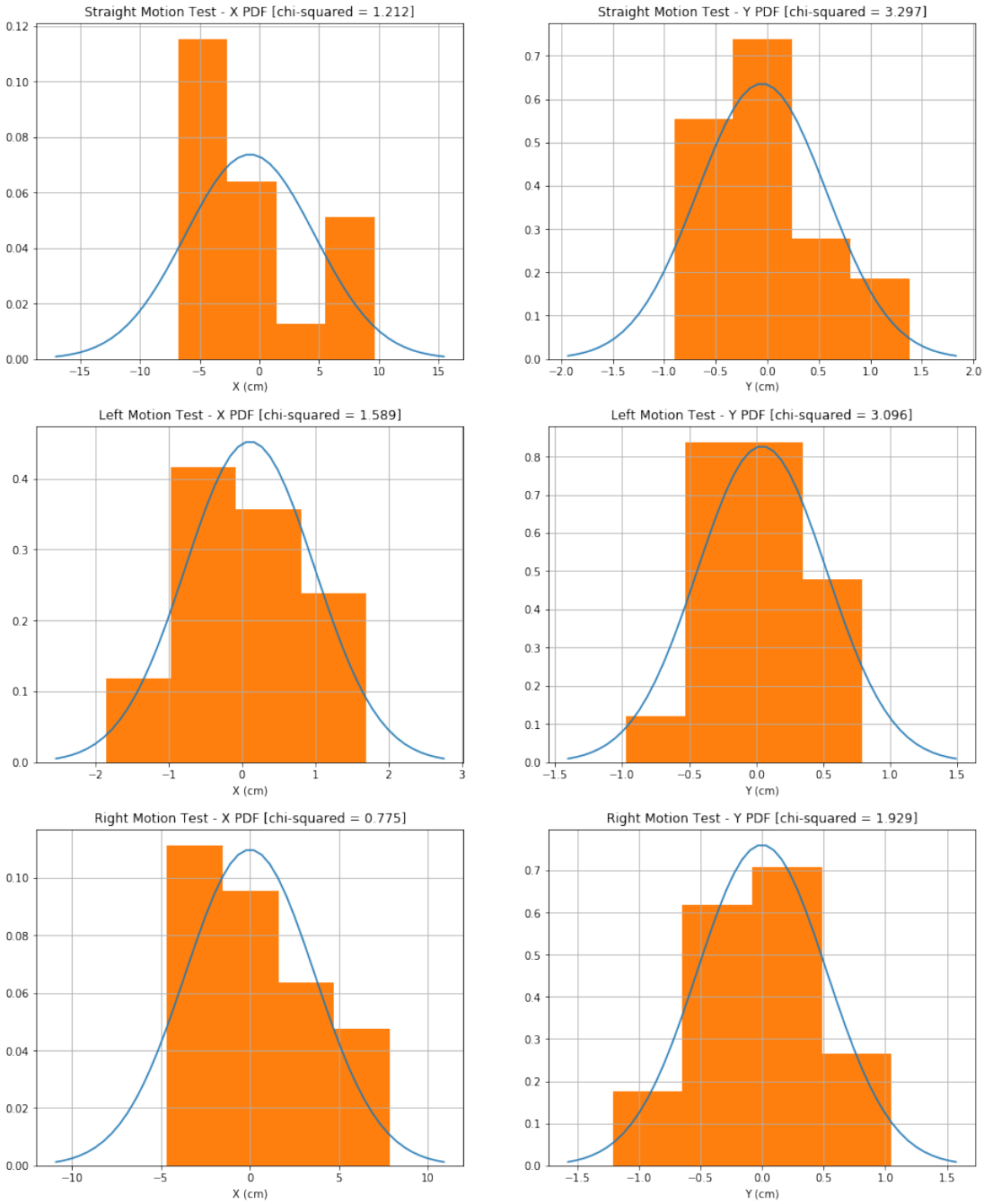


Figure 1.9: Chi-squared Test with PCA inliers

9. Accuracy and Precision

- Under ideal scenarios everything should work fine and the final pose of

the experiment run should match the pose calculated by theory. But in real world a lot of errors creep in.

- For instance, following can be the list of sources which can introduce errors:
  (a) Improper initialization of experimental run (order of error can be in millimeters).
  (b) Unsymmetrical actuation of motors leading to incorrect final pose (order of error can be in centimeters).
  (c) Human factors affecting measurement, for instance, parallax errors while reading a ruler (order of error can be in millimeters).
  (d) Wheel slippage while the experimental run. (order of error can be in millimeters).

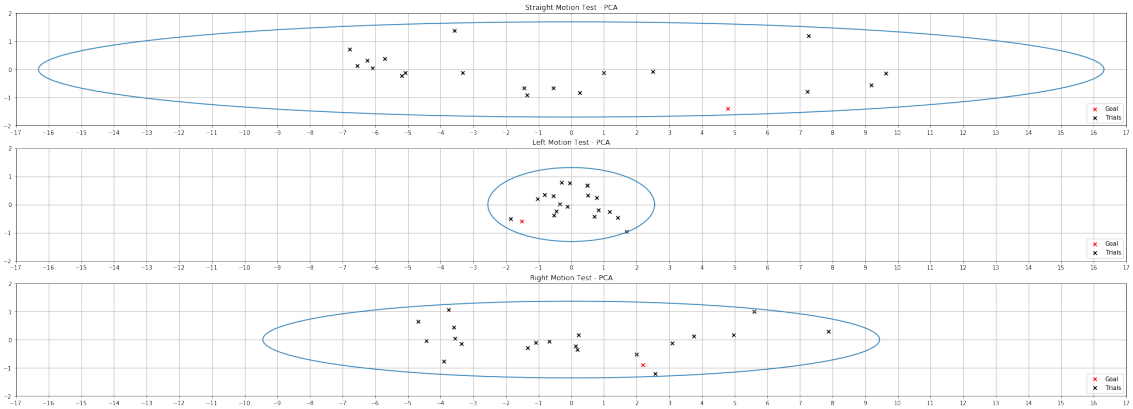


Figure 1.10: Accuracy and Precision Test

Test	Precision (cm)	Accuracy (cm)
Straight Motion	16.4	7.3
Left Motion	2.9	1.9
Right Motion	9.5	3.8

Table 1.1: Accuracy and Precision Test Results

10. Conclusion

- First we conducted the chi square test for the data without conducting PCA.
- Next, we conducted the PCA and the chi-square test.
- We found that the chi-square value decreased in most cases, which is favorable.

2

Camera Calibration

2.1 Aim

To calibrate camera using OpenCV python and chessboard images.

2.2 Procedure

1. In some cases the camera can be moved while in others the image plane can be changed.
2. In our case, the camera was fixed on the table, so that it can not move and create distortion in the image.
3. First we take a 45 chessboard image.
4. The images of the board are taken from the provided camera under different orientations.
5. Some of the example images are shown in the figures below.
6. These pictures were than processed using OpenCV for camera parameters.

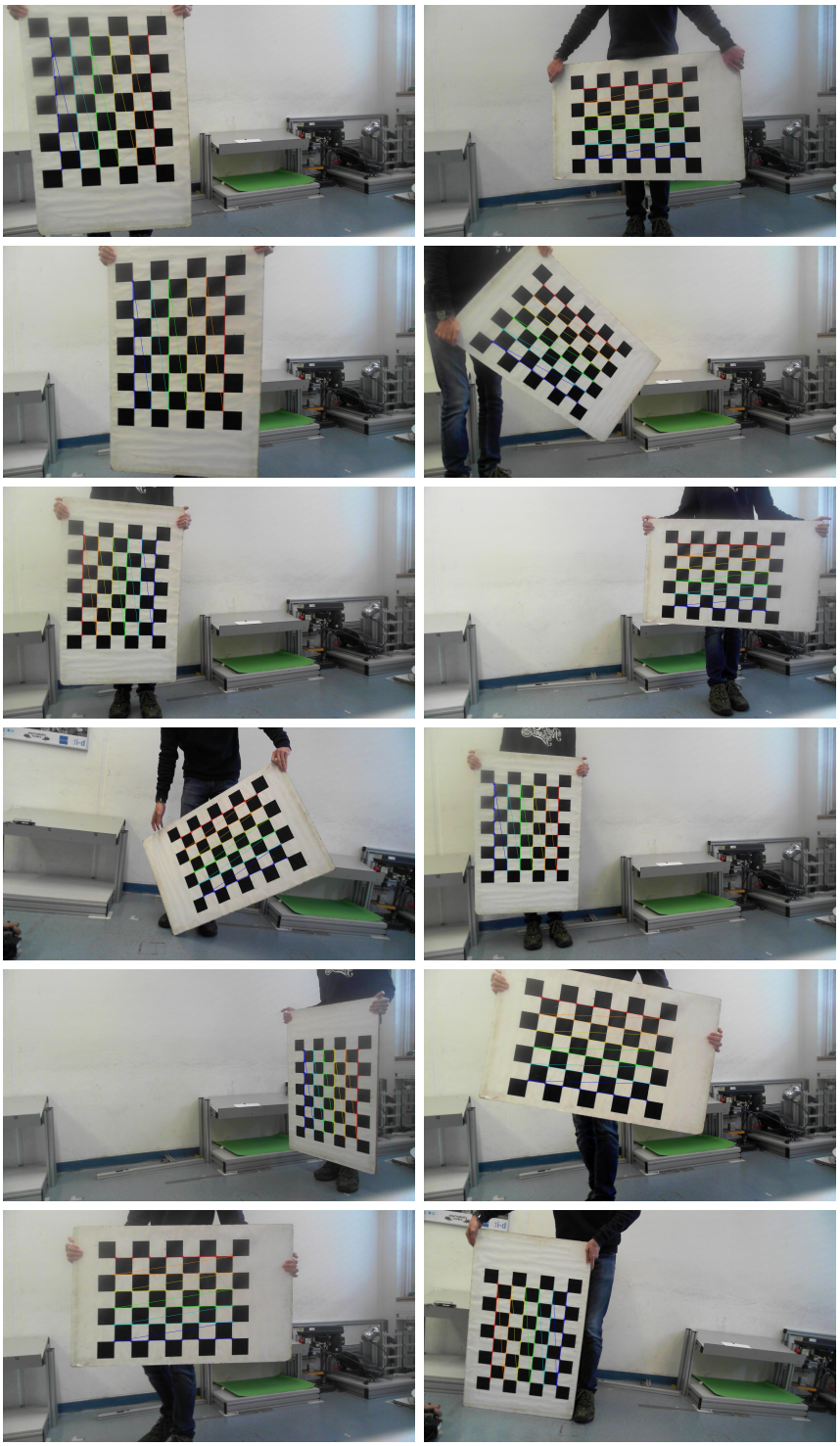


Figure 2.1: Different chessboard labeled points view

7. The camera used is a Microsot LifeCam. It was mounted on the stand to make sure it is fixed in its position and avoids shaking, which could also cause distortion.

Figure 2.2: Microsoft Camera

2.3 Observations

1. There are two major types of distortions.
   (a) Radial distortion: Straight lines will appear curved.

   (b) Tangential distortion: Lense is not aligned perfectly parallel to the imaging plane. So some areas in image may look nearer than expected.

2. Apart from distortions a few more information is needed for the cameras.

   (a) Intrinsic paramaters
   These are specific to a camera. It includes information like focal length (f_x, f_y), optical centers (c_x, c_y) etc. It is also called camera matrix.

   (b) Extrinsic parameters
   These parameters corresponds to rotation and translation vectors which translates a coordinates of a 3D point to a coordinate system.

3. The experiment was carried out on 45 images.

4. The variation in the images can be provided by using different positions of monitor and camera as shown in the images below.

5. We rotated the chessboard and varied the distance to get different images.

6. After working on our own images we searched on the internet for other possible images and techniques using which distortion can be corrected.

7. The other two approaches we found useful were:

8. Mounting the camera on fixed point and than taking the pictures like using a laptop taking different views of the images can help for a better estimate of camera matrix. See Figure **??**.

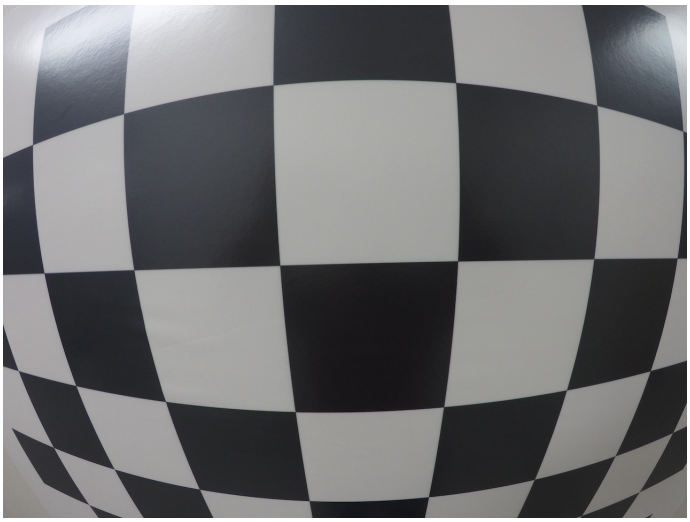


Figure 2.3: Test image [4]

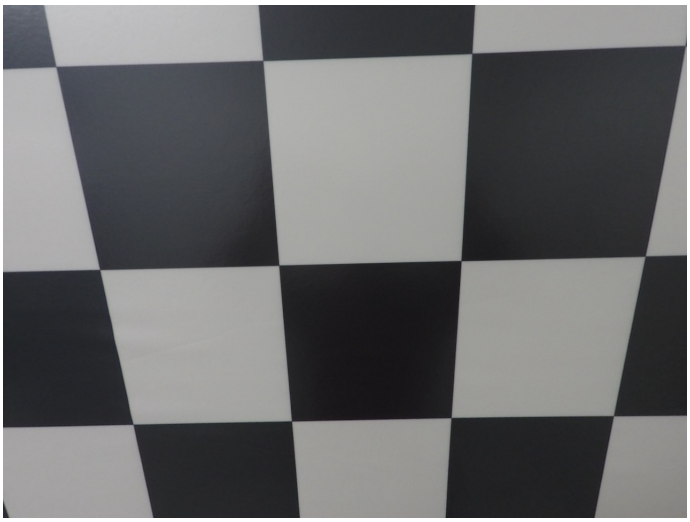


Figure 2.4: Undistorted Image [4]

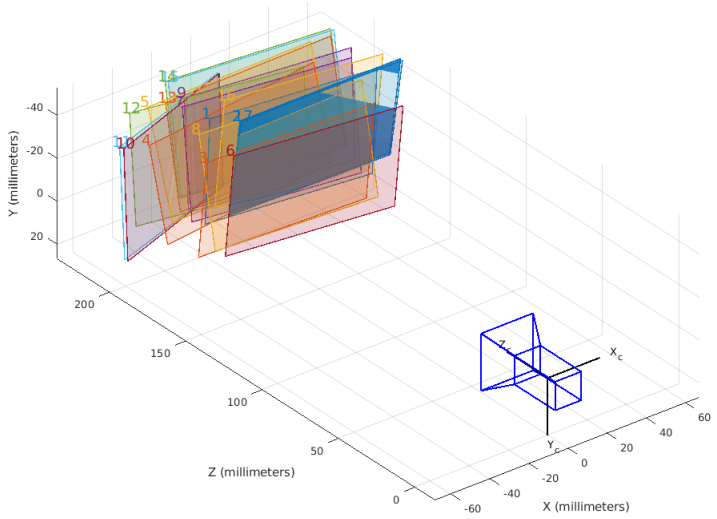


Figure 2.5: Experiment configuration [3]

2.4 Conclusion

- In this experiment we estimated the different parameters of a Microsot Lifecam as described in the 'Experimental Results' section.

2.5 Experimental Results

1. Intrinsic Parameters
   Intrinsic Parameters are specific to a camera. They include information like focal length (f_x, f_y) and optical centers (cx,cy). The focal length and optical centers can be used to create a camera matrix, which can be used to remove distortion due to the lenses of a specific camera. The camera matrix is unique to a specific camera, so once calculated, it can be reused on other images taken by the same camera. It is expressed as a 3x3 matrix.

   (a) Skew = 0.0
   Skew between two axis. The simplest 3D case of a skew coordinate system is a Cartesian one where one of the axes (say the x axis) has been bent by some angle , staying orthogonal to one of the remaining two axes [5]

   (b) Focal length = [1436.53 1439.91]
   The distance between the center of a lens and its focus

(c) Principal point = [902.97 610.26]
The point on the image plane which is at the base of the perpendicular from the center of the lens

(d) Radial distortion = [-0.005 0.101 -0.135]
Straight lines appears curved, this coefficient while undistortion will convert back to straight

(e) Tangent distortion = [0.008 -0.011]
Due to alignment issues of lense and image plane, some points seems nearer than actual, while undistortion this coefficient will re-translate them back to the original position

2. Extrinsic Parameters
Extrinsic parameters corresponds to rotation and translation vectors which transform the coordinates of a 3D point in the world to the corrdinates of a 2D point in the image.

(a) Rotation Vectors = [3×1×45 double]
For each respective image, this vector holds the rotation vector required to transform from the 3D camera frame to the 3D world frame.

(b) Translation Vectors= [3×1×45 double]
For each respective image, this vector holds the translation vector required to transform from the 3D camera frame to the 3D world frame.

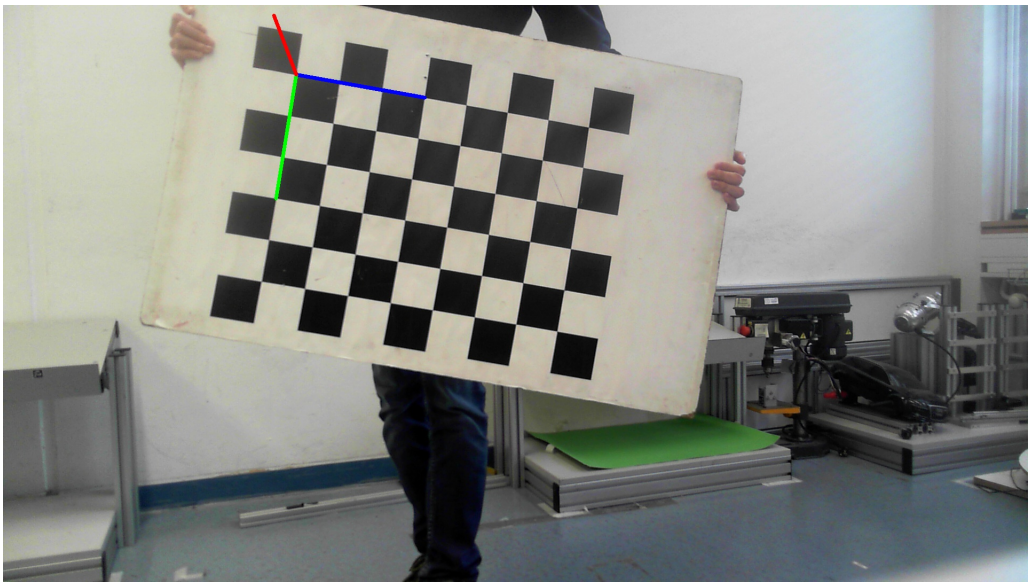


Figure 2.6: World frame

The theoretical explanation of calculation for extrinsic parameters is given as follows:

$$ {}^{W}_{m}T = {}^{W}_{C}T \; {}^{C}_{m}T \tag{2.1} $$

Where

${}^{C}_{m}T$ - transformation matrix denoting marker with respect to camera

${}^{W}_{C}T$ - transformation matrix denoting camera with respect to world

W_mT - transformation matrix denoting marker with respect to world

The measurements obtained by the camera would be in 'the marker with respect to camera' frame. We need to convert the measurements for marker with respect to the world frame. To do so, we need transformation for camera with respect to the world frame.

3. Accuracy of Estimations

   (a) Mean Reprojection error = 0.192

   (b) Reprojected points = [1×2×48 double]

4. Calibration settings

   (a) Number of patterns = 45

   (b) World points = [48×2 double]

   (c) World unit = millimeters

   (d) Number of radial distortion coefficients = 3

3

Measuring the Accuracy and Precision of a KUKA youBot Arm

3.1 Objective

Estimation of the accuracy and precision of a KUKA youBot arm by performing 3 different types of placing tasks.

3.2 Materials

- KUKA robot arm
- Computer for controlling the robot arm
- 3 objects of different sizes but identical geometry
- ROS software
- Provided files from the TA
- ArUco markers
- Camera for measurement

3.3 Setup

- Launch the following commands to control the arm.
  - `roscore`
  - `roslaunch youbot_driver_ros_interface youbot_driver.launch`
  - `roslaunch youbot_moveit move_group.launch`
  - `roslaunch youbot_placing_experiment youbot_placing.launch`
- To recover the data observed by the camera, make sure to connect your device to the laboratory network as the data is published over that network by making use of the ZMQ publisher.
- A subscriber is provided to capture the data from the camera which can be executed by running the following command:

  `python <path to downloaded script>/<script name>.py <marker ID>`
- Make sure to use object with same marker ID as the youbot arm number.
- While repeating the experiments make sure that the orientation of the object are same throughout and are as depicted in the images attached ahead.

3.4 Procedure

1. Place the robot arm in a pre-grasp pose as shown in the figure.

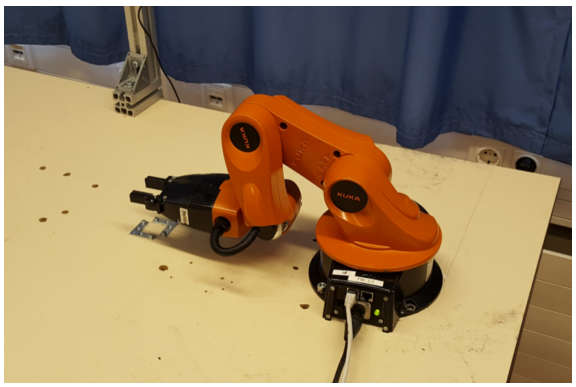


Figure 3.1: pre-grasp pose

2. Grip the SMALL brick and place it to the left. Store the data generated by the camera and repeat this 20 times in total. Now take the brick and use the gripper to place it to the right. Capture the data generated by the camera. Finally, grip the brick and place it in from of the robot. Repeat this 20 times in total and capture the data generated by the robot.

3. Repeat the above step for MEDIUM and LARGE bricks as well. By the end of this step, you will have 180 data files containing the x and y positions, and the orientations captured by the camera.

4. Visualize the data by plotting it.

3.5 Sources of Error

3.5.1 Initial Remarks

- The robot arm might not function properly, retraction of the arm might disturb the brick and hence can affect the readings obtained by the camera.

- The arm might not release the object properly due to unsymmetrical actuation of the gripper pincers. This may lead to skewed orientation and position readings of the brick base.

- The platform on which the experiment is being conducted on may suffer from disturbances and hence can affect the readings measured.
- The base of the brick at initial position might not be stable which might lead to unstable pick configuration.

3.5.2 Uncertainties

- How much does changing the shape and mass of an object affect the final placing pose? Is this effect statistically significant? If the effects are indeed significant, what might have caused them?
  - Changing the mass of the object might change the final pose of the robot arm. For our set of readings, the data appears to be a bit varied for the case of medium brick. This might be due to the reason of poor finish of the base surface of the brick resulting in an unstable base. Surface irregularities present on the drop point might also play a role in the disturbed readings.
  - Changing the shape of object will result in it being picked up by the gripper from different points. This can also affect how the object is deposited back onto the platform.
- If we are to use a single camera measurement for determining the final object pose instead of multiple filtered measurements, is the effect on the final pose estimates significant?
  - The essence of scientific experimentation is basically to achieve the ability of repeating experiments with same results. Hence, multiple measurements are needed to get a more better estimate of the position and orientation. Here, several things which can go wrong if only a single camera measurement is recorded. For instance-
    - Spurious noise in the camera readings is minimized when a number of readings are recorded

* Disturbances, is any on the platform of operation are allowed to die out when multiple readings are measured.

3.6 Experimental Results

3.6.1 Visualization of final object poses

From Figure **??**, we can observe the placing performance while handling the small, medium and large objects. Note that the manipulatoin test for the large object shows greater precision.

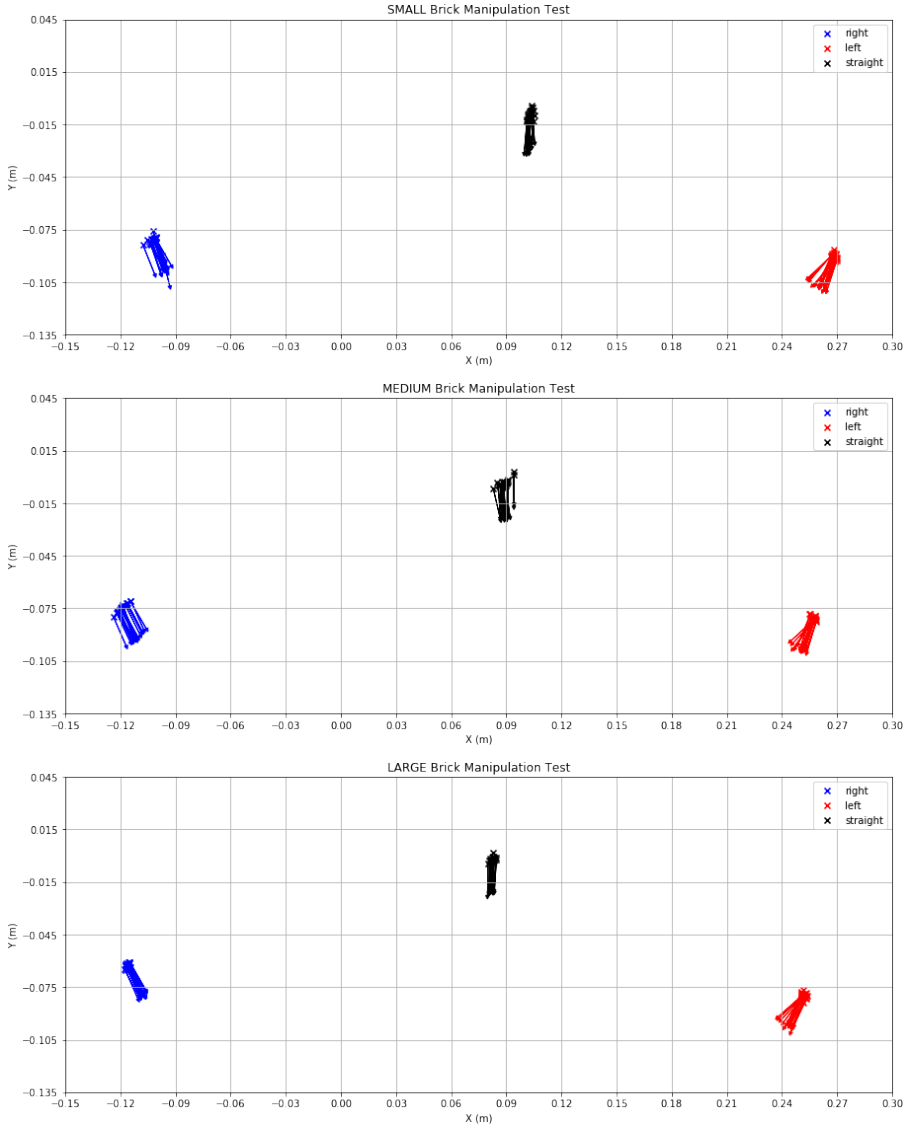


Figure 3.2: Computed final poses

3.6.2 Outlier Detection

First, PCA was applied to the data to normalize it. Then, an ellipse was drawn with 3 times the standard deviation of x as one of the ellipse axis and 3 times the standard deviation of y as the other axis. Note that this number (3) represents a 97% confidence in data robustness.

From Figure **??**, we deemed as an outlier every data point that lies outside the ellipse.

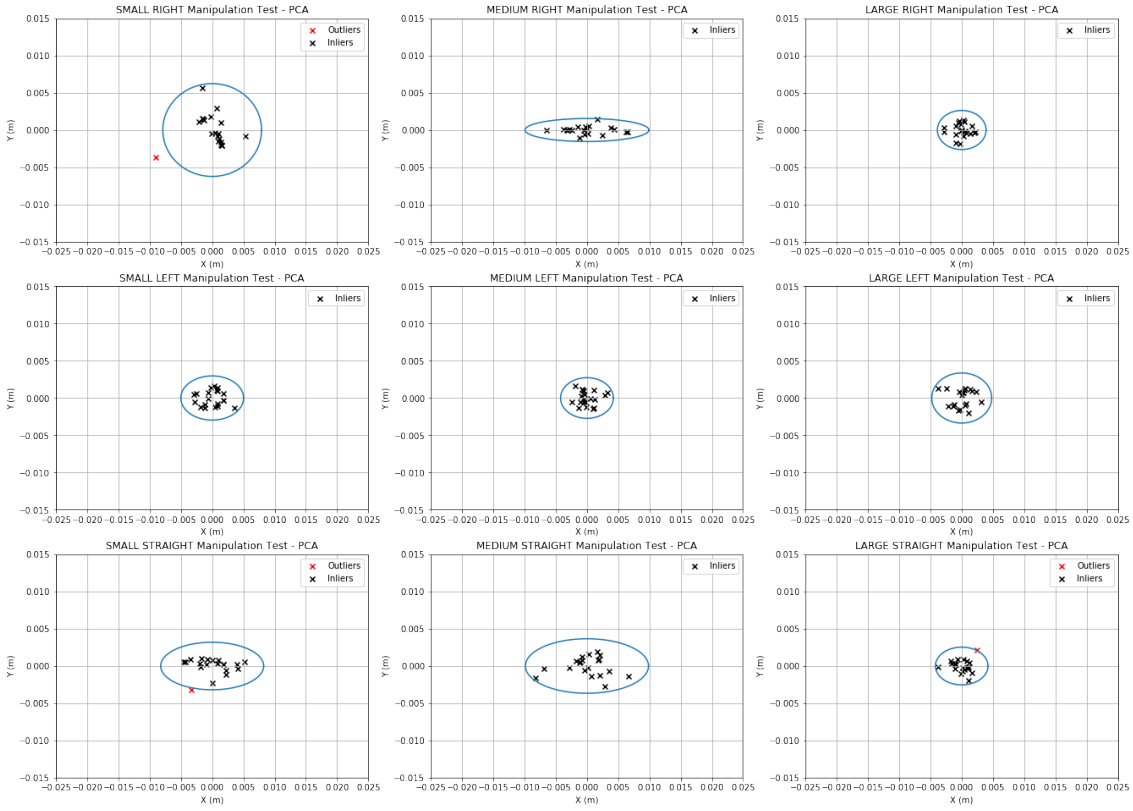


Figure 3.3: Outliers and Inliers

3.6.3 Accuracy and Precision

In this section, we used the expected values from the lab manual to measure our accuracy.

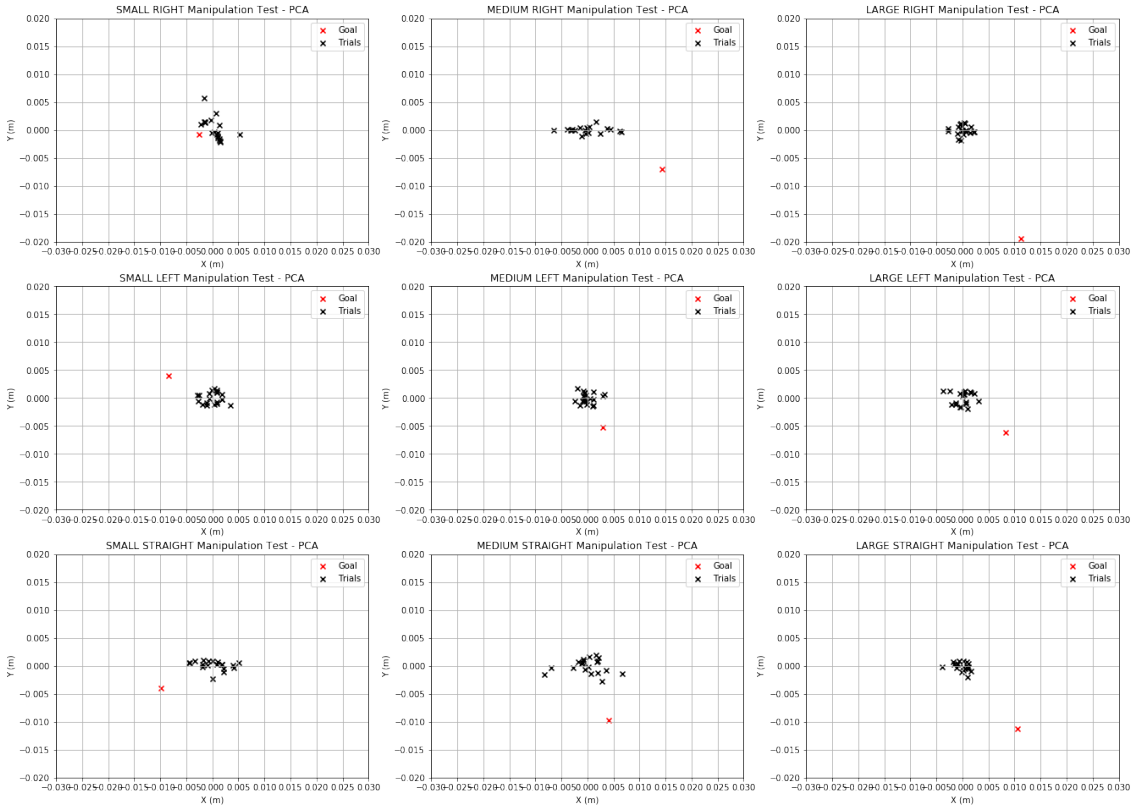


Figure 3.4: Accuracy and Precision

Test Mode	Precision (m)	Accuracy (m)
Straight	0.009	0.011
Left	0.006	0.009
Right	0.010	0.004

Table 3.1: Small Brick Test - Accuracy and Precision

Test Mode	Precision (m)	Accuracy (m)
Straight	0.011	0.011
Left	0.005	0.006
Right	0.010	0.016

Table 3.2: Medium Brick Test - Accuracy and Precision

Test Mode	Precision (m)	Accuracy (m)
Straight	0.005	0.015
Left	0.006	0.010
Right	0.005	0.022

Table 3.3: Large Brick Test - Accuracy and Precision

3.6.4 Chi-squared test

Figures ??,??,?? show the chi-squared test applied to both x and y dimensions of the data for each respective object size. The number of bins selected for the histograms during this test was 4. This value was chosen experimentally.

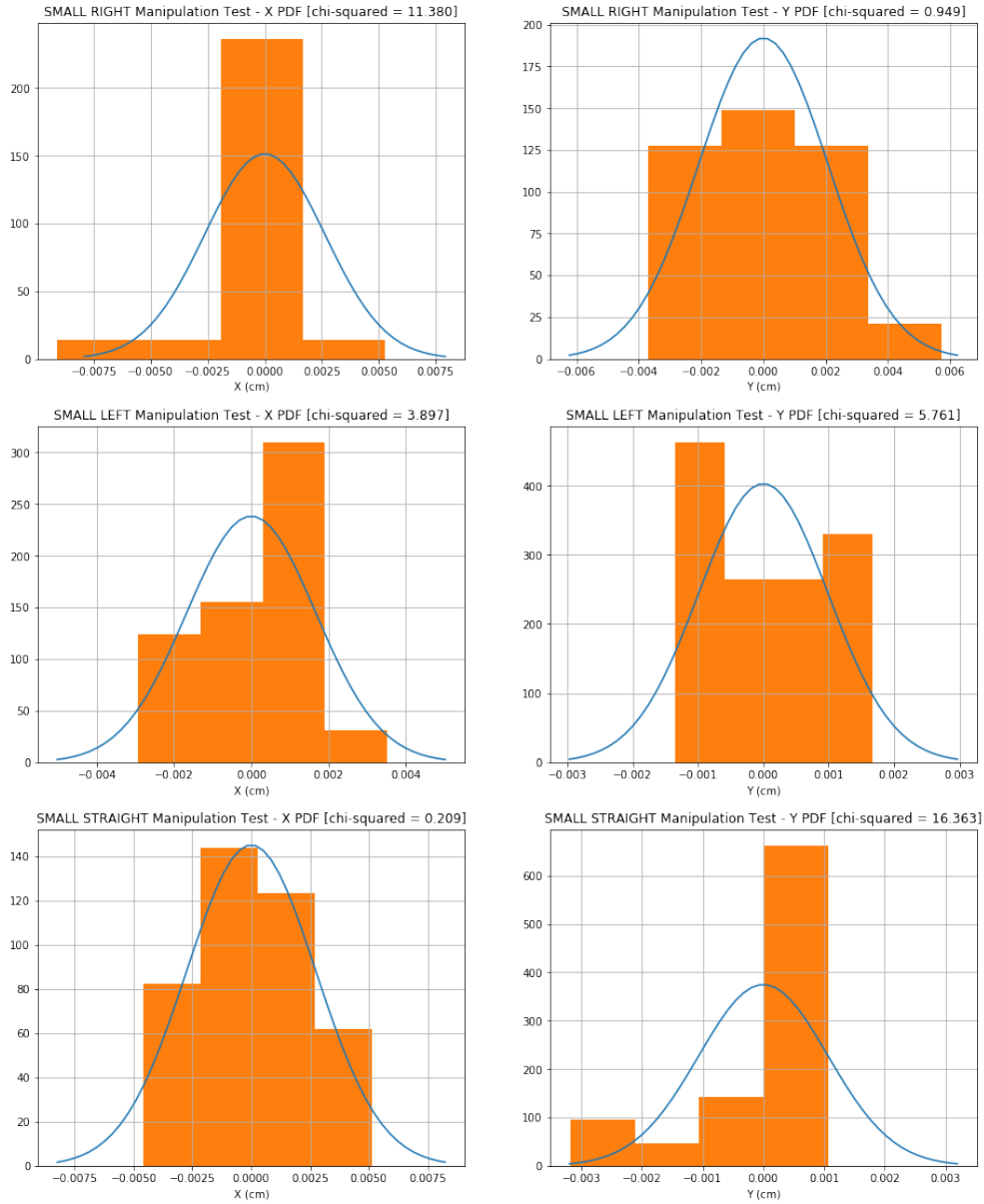


Figure 3.5: Small Brick Chi-squared Test

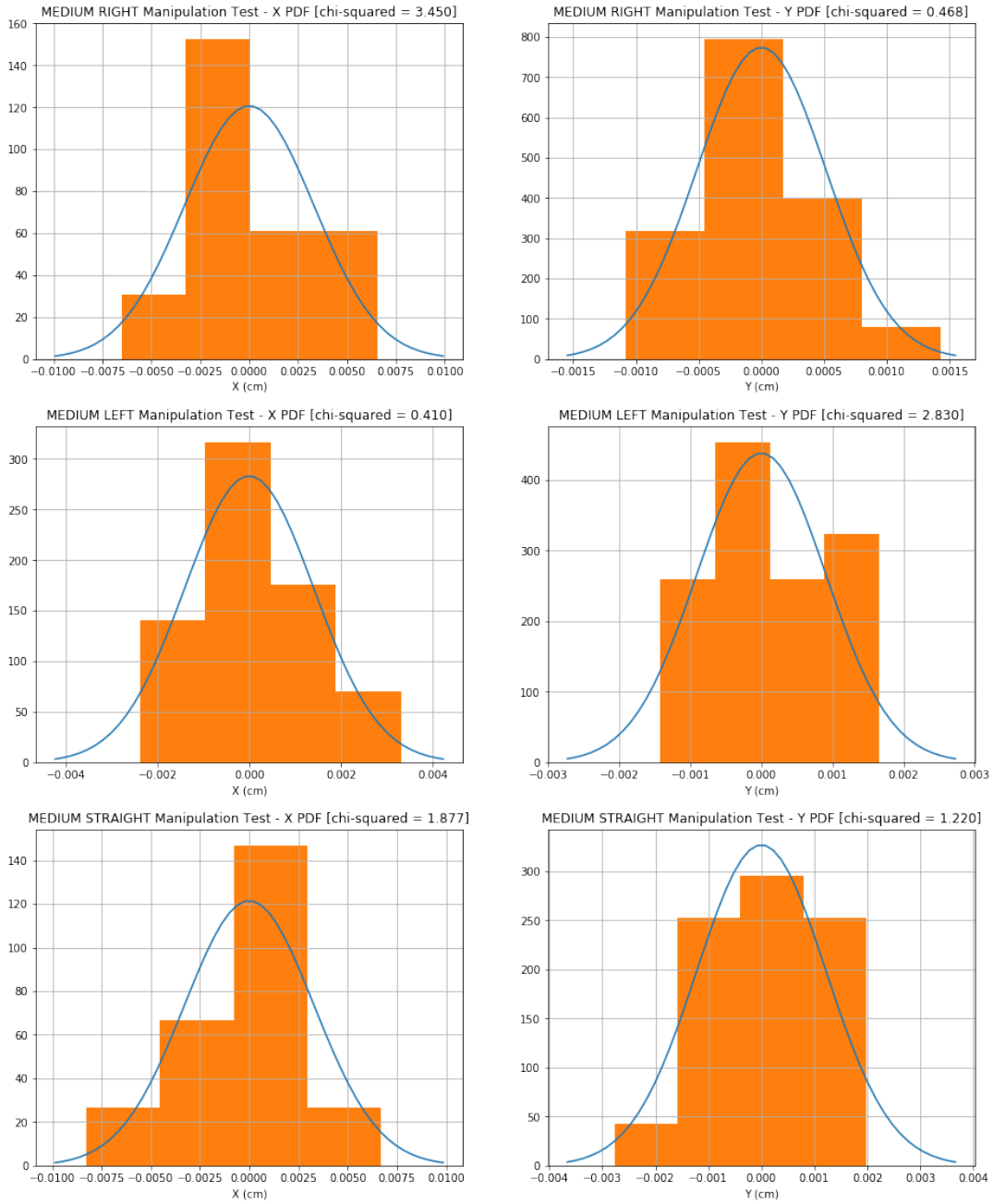


Figure 3.6: Medium Brick Chi-squared Test

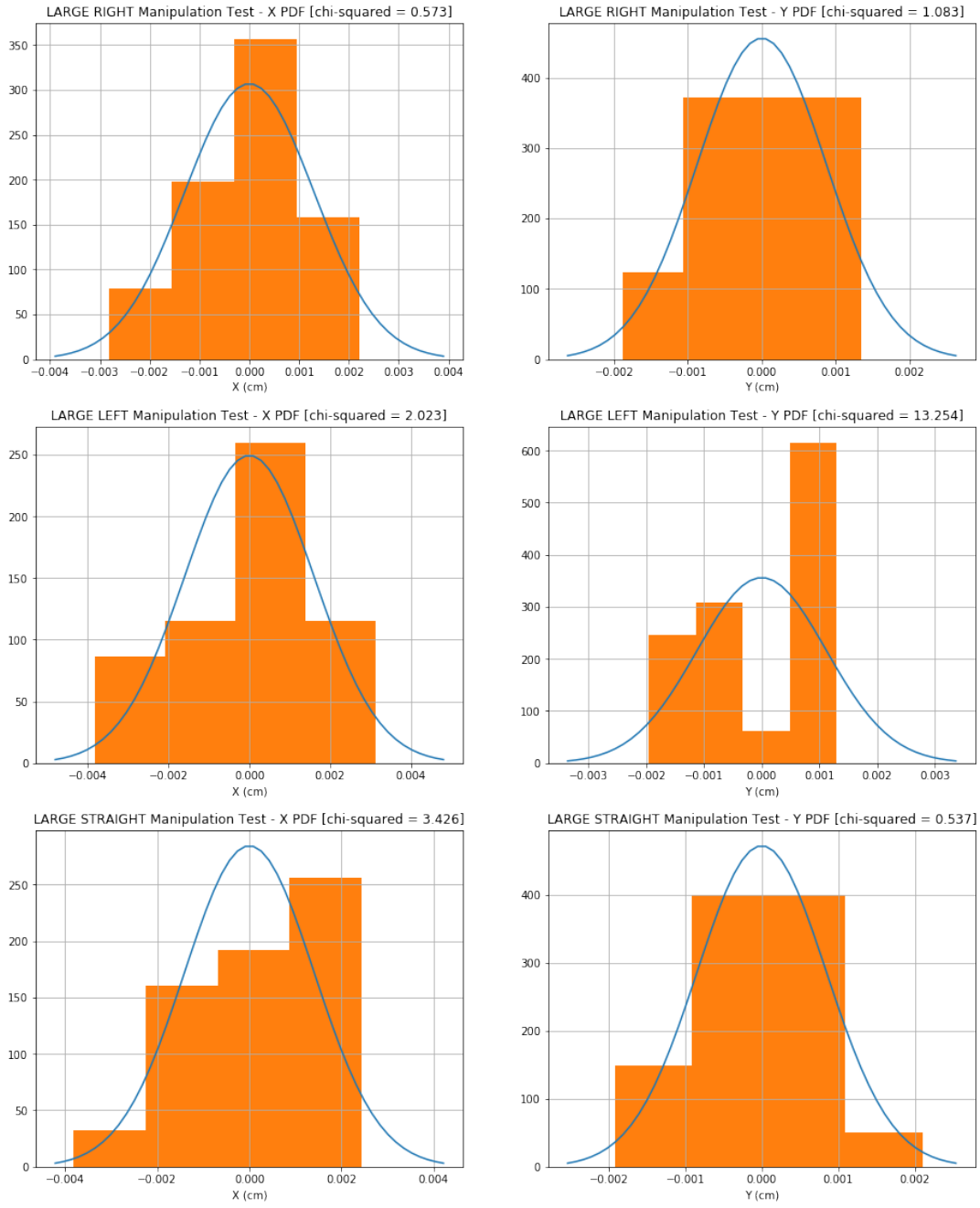


Figure 3.7: Large Brick Chi-squared Test

3.6.5 Welch t-test

Figure **??** shows the t-test applied to all pairs of object sizes with respect to the x, y and theta dimensions of the data.

```
For X values
=============
Between  SMALL  &  MEDIUM  sizes in RIGHT mode:  3.864422873078214e-15
Between  SMALL  &  MEDIUM  sizes in LEFT mode:  2.677746517155014e-15
Between  SMALL  &  MEDIUM  sizes in STRAIGHT mode:  2.2198663378775843e-14
Between  SMALL  &  LARGE  sizes in RIGHT mode:  1.260274822734434e-14
Between  SMALL  &  LARGE  sizes in LEFT mode:  7.560566209398371e-15
Between  SMALL  &  LARGE  sizes in STRAIGHT mode:  3.9692616044392316e-14
Between  MEDIUM  &  LARGE  sizes in RIGHT mode:  2.5355046276546106e-14
Between  MEDIUM  &  LARGE  sizes in LEFT mode:  1.054878953634492e-14
Between  MEDIUM  &  LARGE  sizes in STRAIGHT mode:  3.8561976967574956e-15

For Y values
=============
Between  SMALL  &  MEDIUM  sizes in RIGHT mode:  3.5292063527126097e-15
Between  SMALL  &  MEDIUM  sizes in LEFT mode:  1.108223557564611e-14
Between  SMALL  &  MEDIUM  sizes in STRAIGHT mode:  2.2835497806861203e-14
Between  SMALL  &  LARGE  sizes in RIGHT mode:  2.831693411484461e-14
Between  SMALL  &  LARGE  sizes in LEFT mode:  1.5027638252663934e-14
Between  SMALL  &  LARGE  sizes in STRAIGHT mode:  1.5122772833803392e-14
Between  MEDIUM  &  LARGE  sizes in RIGHT mode:  2.69698179660481e-14
Between  MEDIUM  &  LARGE  sizes in LEFT mode:  2.8163885460863053e-14
Between  MEDIUM  &  LARGE  sizes in STRAIGHT mode:  1.4410526534424964e-14

For THETA values
=============
Between  SMALL  &  MEDIUM  sizes in RIGHT mode:  7.406105878256925e-14
Between  SMALL  &  MEDIUM  sizes in LEFT mode:  1.6874845359403188e-13
Between  SMALL  &  MEDIUM  sizes in STRAIGHT mode:  6.479599164818866e-14
Between  SMALL  &  LARGE  sizes in RIGHT mode:  4.082627239858835e-14
Between  SMALL  &  LARGE  sizes in LEFT mode:  1.839183608001471e-13
Between  SMALL  &  LARGE  sizes in STRAIGHT mode:  3.64640819661784e-14
Between  MEDIUM  &  LARGE  sizes in RIGHT mode:  6.151256669460595e-14
Between  MEDIUM  &  LARGE  sizes in LEFT mode:  3.412674060207389e-14
Between  MEDIUM  &  LARGE  sizes in STRAIGHT mode:  3.5923995660843075e-14
```

Figure 3.8: Welch T-Test

3.7 Conclusion

- From Figure **??**, we can conclude that the KUKA youBot arm is accurate in a 2.2 cm radius and precise in a 1.1 cm radius. Additionally, the small object provided the highest accuracy and the large object provided the highest precision. The latter serves as proof to our previous observation from Figure **??**. The reason behind this particular behavior is the difference in mass. While the reduced mass of the small object reduces the torque that the KUKA arm has to exert to move the object, hence increasing accuracy, the increased mass of the large object increases the momentum of the KUKA arm as it moves to place the object, hence increasing precision.

- From Figure **??**, **??** and **??**, we can conclude that the distribution of the final object poses follow a Gaussian distribution because most chi-values are below 7.81. This means that the difference across regular and normal data isn't significant at the 5% confidence level according to the table of critical values of chi-square and the calculated degrees of freedom (3).

- From Figure **??**, we can conclude that object size doesn't affect the final outcome significantly because all t-test values are below 1.96. This means that the difference between the means of the two compared datasets isn't significant at the 5% confidence level according to the table of critical values of t and the calculated degrees of freedom (Inf).

3.8 References

Figure **??** has been taken from the SEE lab manual.

A

Drive tests

Straight drive testing				
Serial No.	Right wheel		Left wheel	
	x values (cm)	y values (cm)	x values (cm)	y values (cm)
1	33.8	69.0	22.2	65.0
2	22.7	68.0	13.0	60.2
3	30.0	68.8	18.7	61.6
4	28.5	69.3	17.5	63.8
5	30.0	64.5	20.5	69.0
6	38.1	67.2	25.9	67.8
7	32.5	68.7	20.9	64.5
8	36.8	68.0	24.6	68.0
9	26.4	68.0	15.1	62.5
10	33.3	68.8	21.6	66.2
11	34.3	68.4	22.3	66.8
12	34.1	68.6	22.2	67.0
13	34.5	67.0	22.1	67.8
14	34.1	68.7	22.0	66.3
15	36.6	67.7	24.2	68.0
16	37.5	67.6	24.7	68.3
17	38.1	67.8	25.5	68.2
18	38.1	67.8	25.8	67.7
19	37.5	67.6	25.4	67.5
20	24.4	68.4	22.0	66.3

Right turn testing				
Serial No.	Right wheel		Left wheel	
	x values (cm)	y values (cm)	x values (cm)	y values (cm)
1	48.4	32.9	48.2	46.5
2	50.0	32.9	51.6	45.1
3	48.6	32.6	52.1	45.2
4	49.4	32.6	50.5	44.7
5	48.5	33.7	49.2	46.3
6	49.2	32.6	50.7	44.8
7	46.9	35.7	45.5	47.8
8	48.0	34.4	48.0	47.0
9	48.4	34.0	48.7	45.6
10	48.2	35.7	46.6	48.0
11	44.5	37.1	43.1	49.0
12	47.5	35.5	46.7	47.7
13	48.2	32.2	50.4	44.7
14	49.7	32.0	51.4	44.4
15	46.0	36.4	43.9	48.7
16	44.8	38.1	41.8	50.0
17	48.1	34.2	47.8	46.5
18	48.6	32.9	50.8	45.1
19	47.5	34.0	47.6	47.0
20	46.3	36.0	44.9	48.3

Left turn testing				
Serial No.	Right wheel		Left wheel	
	x values (cm)	y values (cm)	x values (cm)	y values (cm)
1	10.0	47.0	12.1	34.9
2	10.0	47.0	12.4	34.7
3	10.3	47.4	11.4	34.9
4	11.5	47.9	11.3	35.3
5	11.5	48.4	11.6	35.7
6	11.1	48.2	11.0	35.4
7	10.5	48.2	11.0	35.4
8	11.3	47.5	11.7	35.2
9	10.5	48.0	11.4	34.9
10	10.3	47.4	11.3	35.3
11	10.9	47.6	11.7	35.4
12	10.9	47.6	12.0	35.5
13	10.2	48.2	11.0	35.4
14	12.8	48.9	13.2	36.5
15	11.7	48.5	12.5	36.1
16	10.9	47.9	12.2	35.2
17	12.4	48.7	12.5	36.1
18	12.0	48.9	12.3	35.9
19	11.1	48.2	11.0	35.4
20	11.6	48.9	12.3	35.9

B

Camera Calibration - Source Code

```
1
2 #!/usr/bin/env python2
3 # -*- coding: utf-8 -*-
4
5 #%%
6
7 import numpy as np
8 import cv2
9 import glob
10 import matplotlib.pyplot as plt
11 import pickle
12
13 #%%
14
15 # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
16 objp = np.zeros((6*7,3), np.float32)
17 objp[:,:2] = np.mgrid[0:7, 0:6].T.reshape(-1,2)
18
19 # Arrays to store object points and image points from all the images.
20 objpoints = [] # 3d points in real world space
21 imgpoints = [] # 2d points in image plane.
22
23 # Make a list of calibration images
24 images = glob.glob('calib/*.jpg')
25
```

```
#%%
# Step through the list and search for chessboard corners
for idx, fname in enumerate(images):
img = cv2.imread(fname)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Find the chessboard corners
ret, corners = cv2.findChessboardCorners(gray, (7,6), None)

# If found, add object points, image points
if ret == True:
objpoints.append(objp)
imgpoints.append(corners)

# Draw and display the corners
cv2.drawChessboardCorners(img, (8,6), corners, ret)
#write_name = 'corners_found'+str(idx)+'.jpg'
#cv2.imwrite(write_name, img)
cv2.imshow('img', img)
cv2.waitKey(500)

cv2.destroyAllWindows()

#%%

# Test undistortion on an image
img = cv2.imread('calibration_wide/test_image.jpg')
img_size = (img.shape[1], img.shape[0])

# Do camera calibration given object points and image points
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints
    , img_size,None,None)


dst = cv2.undistort(img, mtx, dist, None, mtx)
cv2.imwrite('calibration_wide/test_undist.jpg',dst)

# Save the camera calibration result for later use (we won't worry
    about rvecs / tvecs)
```

```
63 dist_pickle = {}
64 dist_pickle["mtx"] = mtx
65 dist_pickle["dist"] = dist
66 pickle.dump( dist_pickle, open( "calibration_wide/wide_dist_pickle.p",
       "wb" ) )
67 #dst = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
68 # Visualize undistortion
69 f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
70 ax1.imshow(img)
71 ax1.set_title('Original Image', fontsize=30)
72 ax2.imshow(dst)
73 ax2.set_title('Undistorted Image', fontsize=30)
```

Listing B.1: Camera Calibration

1. The images included in the solutions are included from the lecture slides titled ‘Measurement as a process.
2. https://github.com/mhwasil/camera_calibration
3. https://www.mathworks.com/help/vision/examples/evaluating-the-accuracy-of-single-camera-calibration.html
4. https://github.com/udacity/CarND-Camera-Calibration