# Scientific Experimentation and Evaluation

Aaqib Parvez Mohammed
Chetan Sidnal
Mihir Patil
Sushma devaramani

June 6, 2018

# Contents

## List of Figures

## 1 Experiment 1: Manual Motion Observation

**Aim:** Manually measure the observable pose variation for three different constant velocity motions(Straight line, an arc to the left and an arc to the right) of LEGO NXT differential drive robot.

### 1.1 Setup

- The measurement system for the experiment includes a white sheet, a LEGO NXT differential drive robot, two pencils and a scale.
- The **device under test(DUT)** is the LEGO NXT differential drive robot.
- Clamp the sheet on a flat surface.
- Construct a robot equipped with two pencils in the front, such that the start and end positions can be marked. This constitutes the measurement facility.
- Install *Lejos* OS on the robot.
- Program the robot for it to run in three different constant velocity motions.

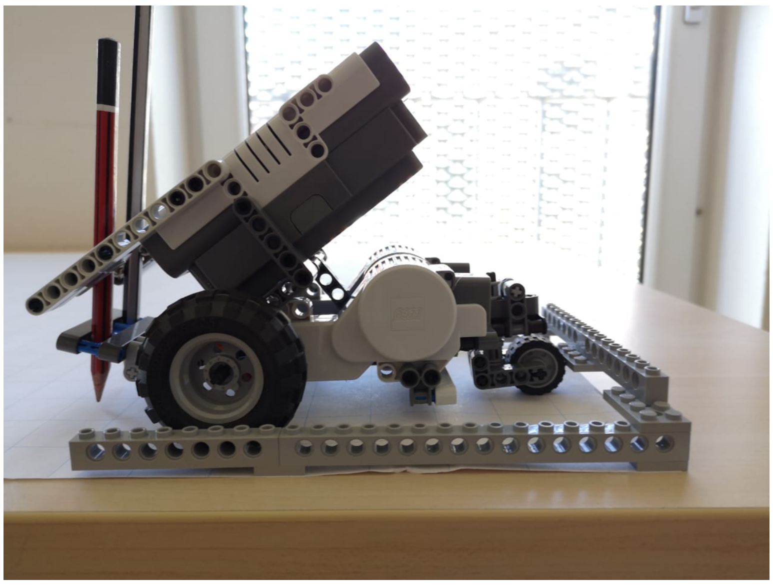


Figure 1: Side view

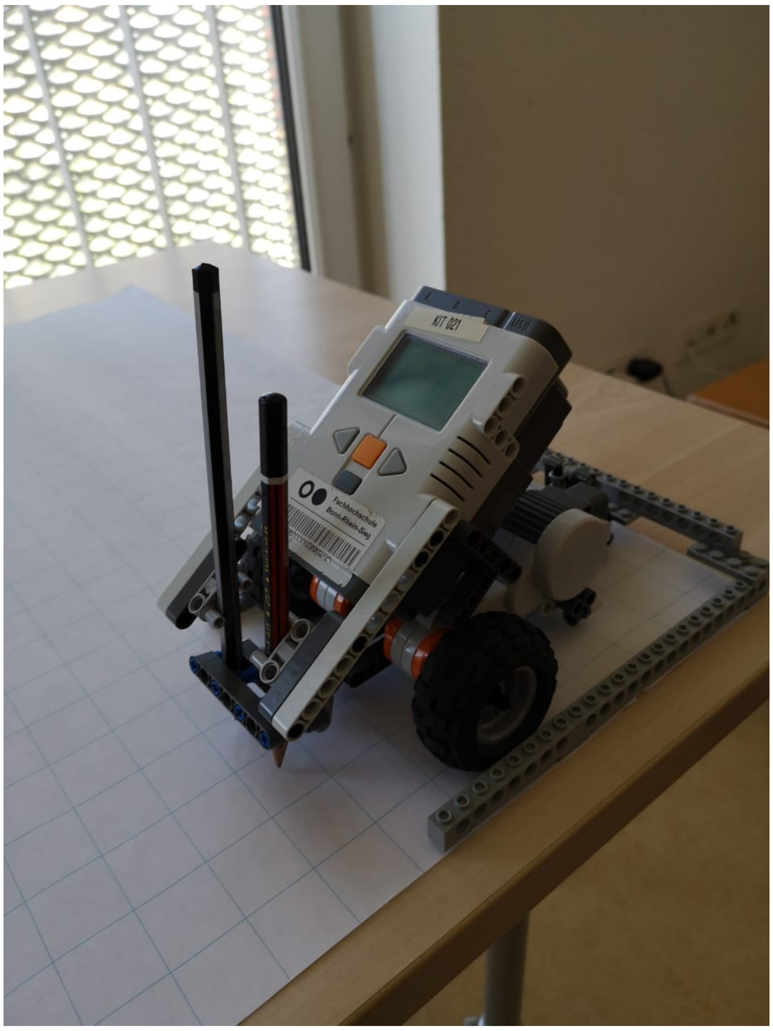


Figure 2: Isometric view

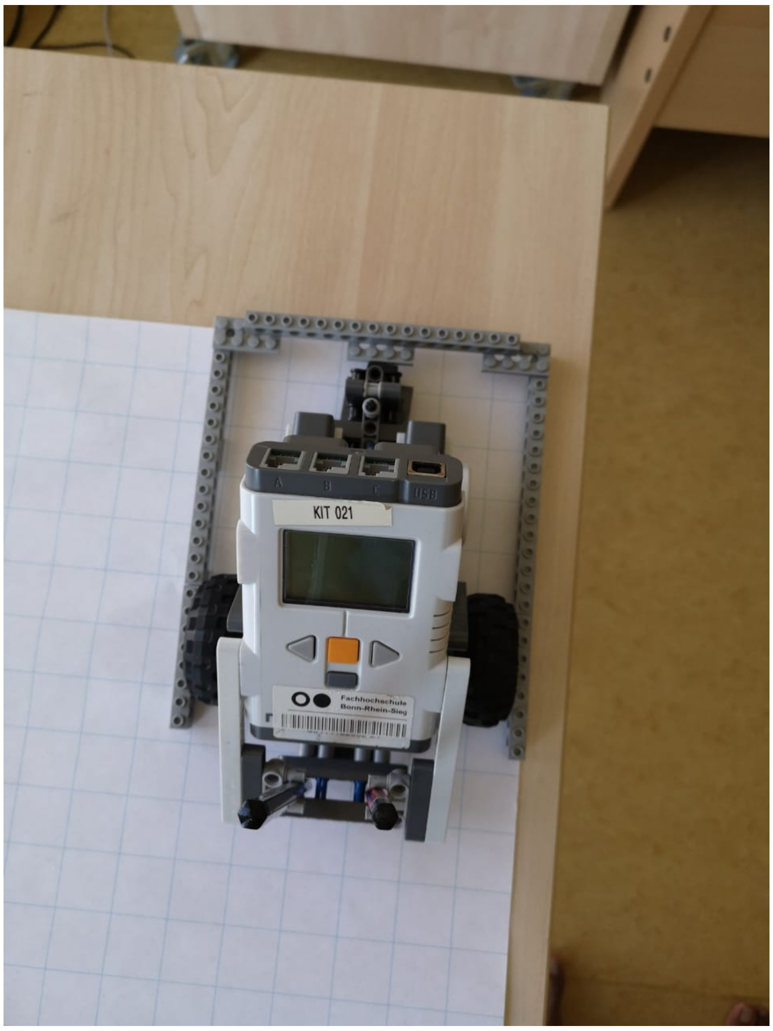


Figure 3: Top view

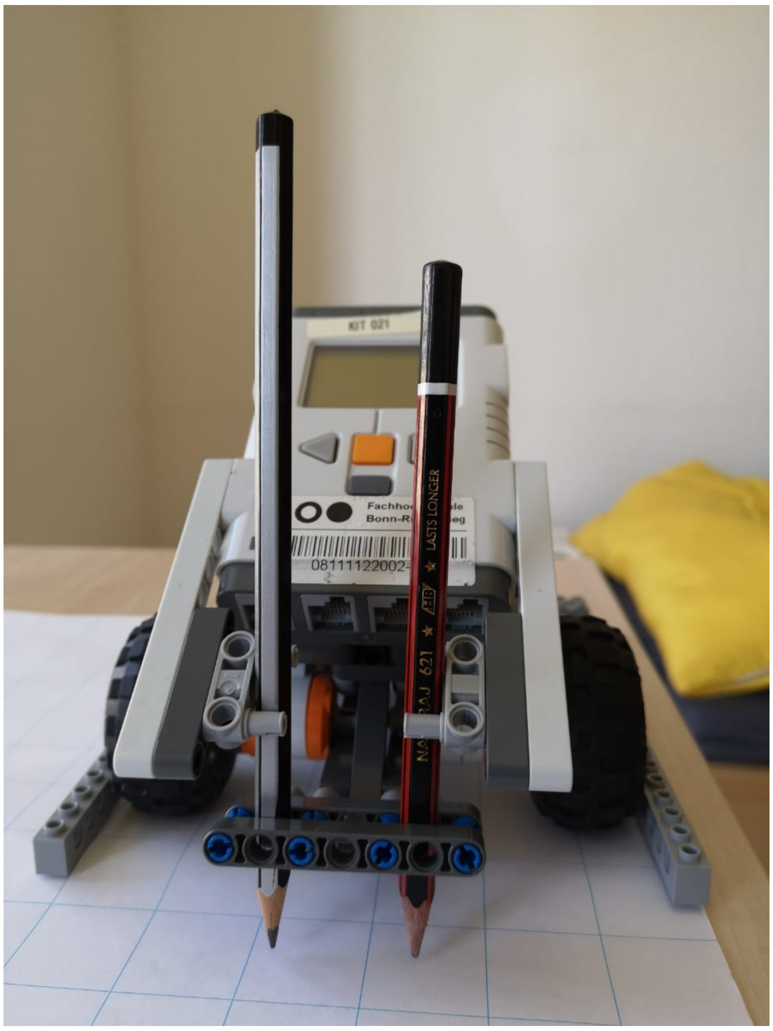


Figure 4: front view

## 1.2 Procedure

- The experimental setup is prepared.
- The starting position of the robot is marked with the help of the points drawn by the pencils.
- The center of the line joining the two points is recorded which will be the starting position of the robot.
- The robot is programmed to run in a straight line forward motion at a constant speed.
- Once the robot stops, the center of the line joining the last points drawn by the pencils will be the stop position.
- The **Measurand** i.e., the relative coordinates of the robot are measured by the difference in the coordinates of the starting position and the stopping position.
- **Measurement System:** Our measurement system consists of robot placed on a cardboard sheet. Two pencils are attached in the front to mark the position. As the robot is moved, the path is traced on the sheet. After reaching the end position, the final pose is marked. The variation in the path is observed and the error in pose is measured.
- The **Measurement result** i.e., the pose variation is obtained by finding the difference between the observed values and their mean.
- Repeat the above experiment with a program to run the robot with constant angular and translational velocities for a fixed time period, such that it describes an arc to the left.
- Repeat the above experiment with a program to run the robot with constant angular and translational velocities for a fixed time period, such that it describes an arc to the right.

## 1.3 Expected problems

- Parallax error.
- Positional errors.
- There might be slipping of wheels.
- The tip of the pencils might break during the motion of the robot.
- The pencils will have to be sharpened as the nibs might get blunt due to continuous drawing of lines which might result in measurement errors.

### 1.4 Expected performance

- The Rotation sensor measures the motor rotations with the accuracy of +/- one degree for one rotation(360 degree). Hence the accuracy in distance traveled for one rotation will be +/- 0.48mm.
- Reference: LEGO NXT: Features & Limitations.
- Assuming experiment distance = 704mm. Expected accuracy is $\pm 2$ and expected precision 704mm is$\pm$2mm.

#### 1.4.1 Reasons

- The slippage of the wheels may cause poor accuracy and precision.
- The internal errors from the DUT may cause poor accuracy but the precision won't be affected significantly, since the DUT won't be changed while doing the measurement.

### 1.5 Execution of the experiment:

- The starting position of the robot was marked using two pencils which were placed on either side of the robot.
- The same points were used as the starting points while repeating the experiment.
- The position of the back wheel was also kept constant by drawing a line and using the same line as reference for future run of experiments.
- The position of the back wheel impacted the outcome of the experiment as a small deviation from the reference line would result in a considerable deviation of the outcome.
- The theta values were calculated by drawing a perpendicular to the line joining the two observed end points and calculating the angle using the formula $tan^{-1}(\frac{y}{x})$.
- Lego NXT 2.0 software was used to program the robot with the following parameters.
  No. of rotations = 4
  Power = 40
- The battery voltage before the experiment = 8.2V.
  The battery voltage after the experimmant = 4.2V.
- The observed data has been recorded in the ".csv" format.

- There was no pre-processing of data required as no outliers were detected.
- Graph has been plotted for all three motions.
  Note : All x,y measurements are in inches

### 1.5.1 Straight

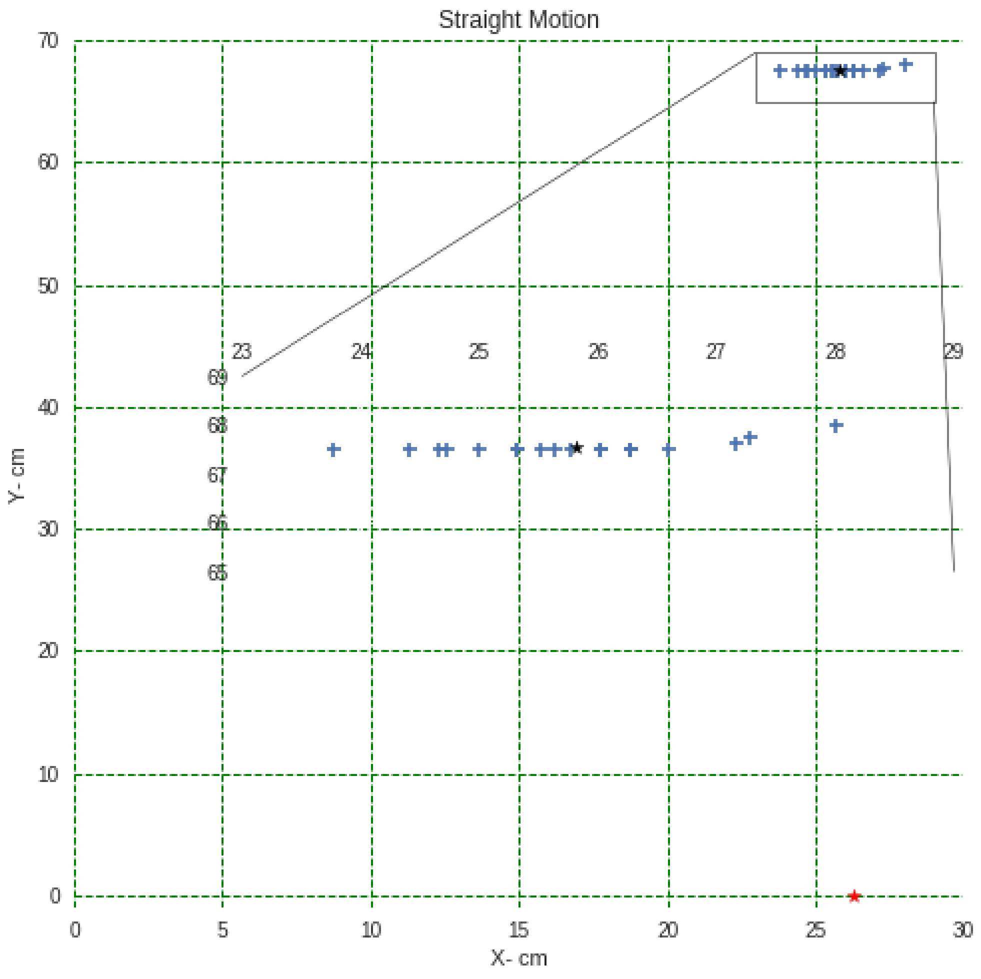


Figure 5: 20 iteration of moving straight

**Fitting the distribution for the obtained data:**
**(X, Y) points for straight line**

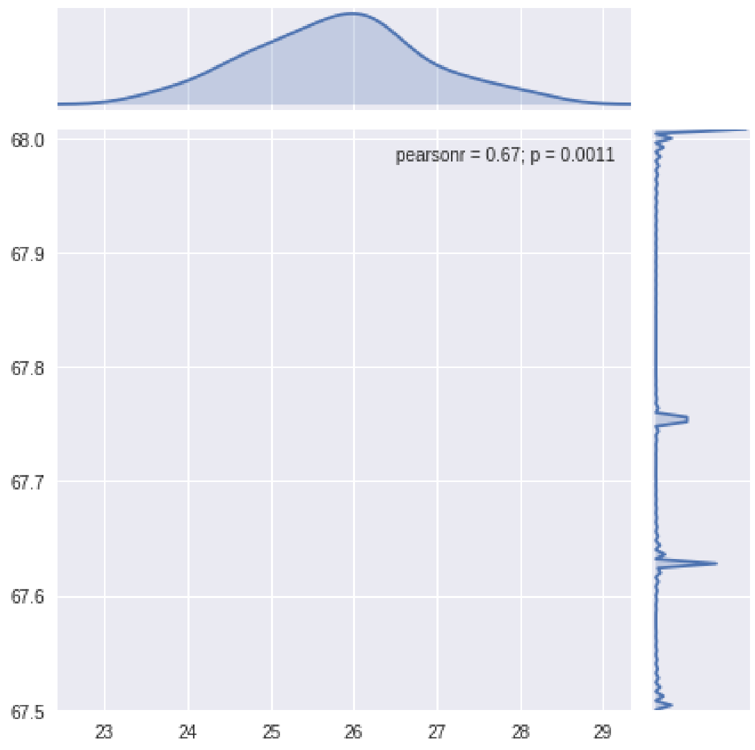


Figure 6: Density plot of (X,Y) for straight line motion

In the above graph, it can be observed that 'X' points fit a gaussian distribution, whereas 'Y' points does not fit any distribution, since they are mostly precise and accurate.

- $Range(x, y) = ([23.77 - 28.01], [67.50 - 68.01])$
- $Mean(x, y) = 25.81, 67.54$
- $\sigma(x, y) = 1.01, 0.12$
- $Accuracy(x, y) = x \pm 0.9, y \pm 0.1$

### 1.5.2 Right

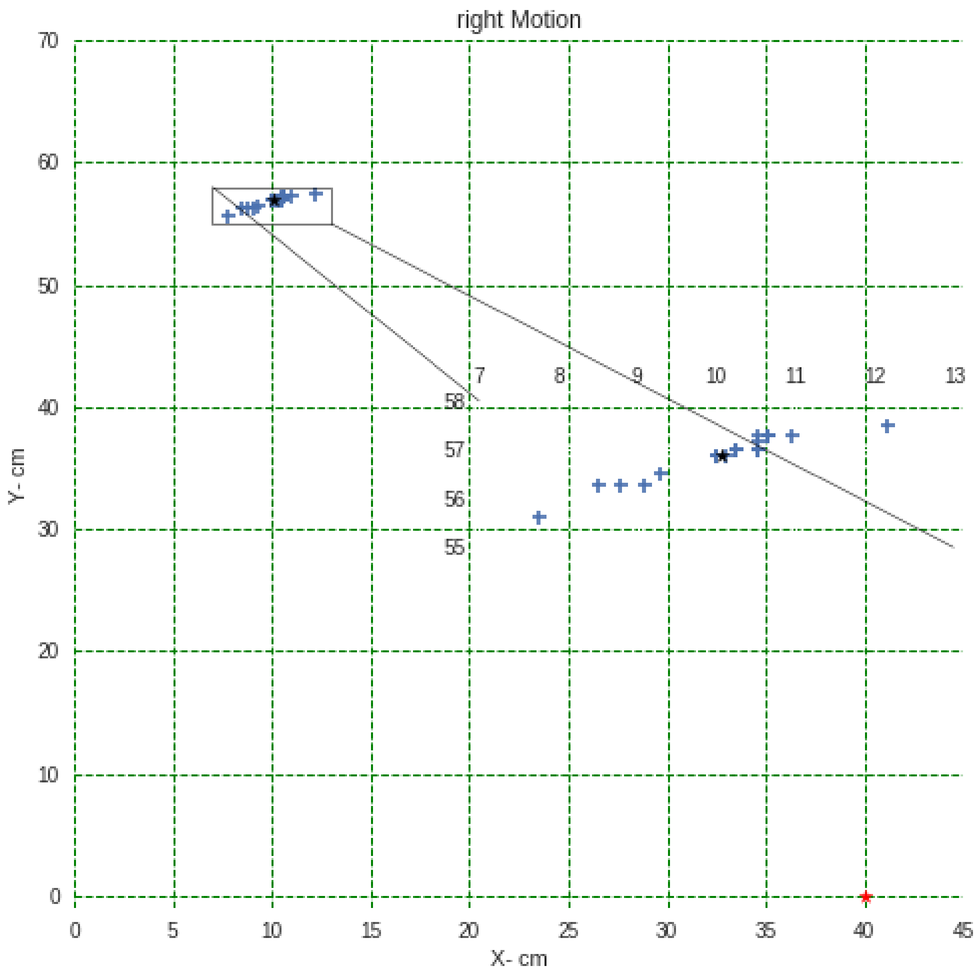


Figure 7: 20 observations of right arc motion

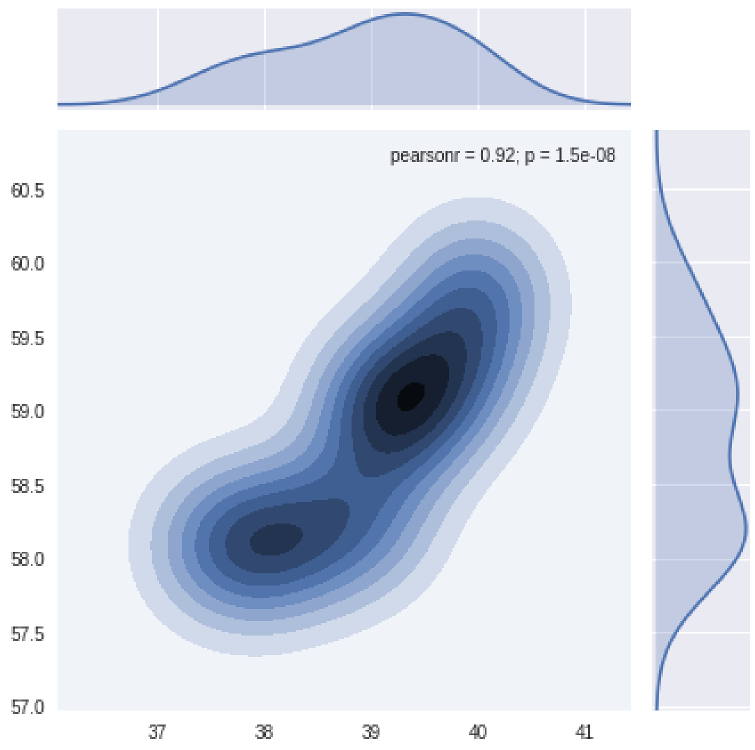


Figure 8: Density plot of (X, Y) points for right arc motion

In the above plot it can be observed that the distribution of X and Y positions for right arc motion is gaussian.

- $Range(x, y) = ([37.50 - 40.00], [58.01 - 59.78])$
- $Mean(x, y) = 38.95, 58.77$
- $Accuracy(x, y) = x \pm 0.5, y \pm 0.5$
- $\sigma(x, y) = 0.81, 0.61$
- $Mean(\theta) = 40.58degree$
- $Accuracy(\theta) = \theta \pm 5deg$
- $\sigma = 1.09deg$
- $range(\theta) = 38.83 - 43.33$

### 1.5.3 Left

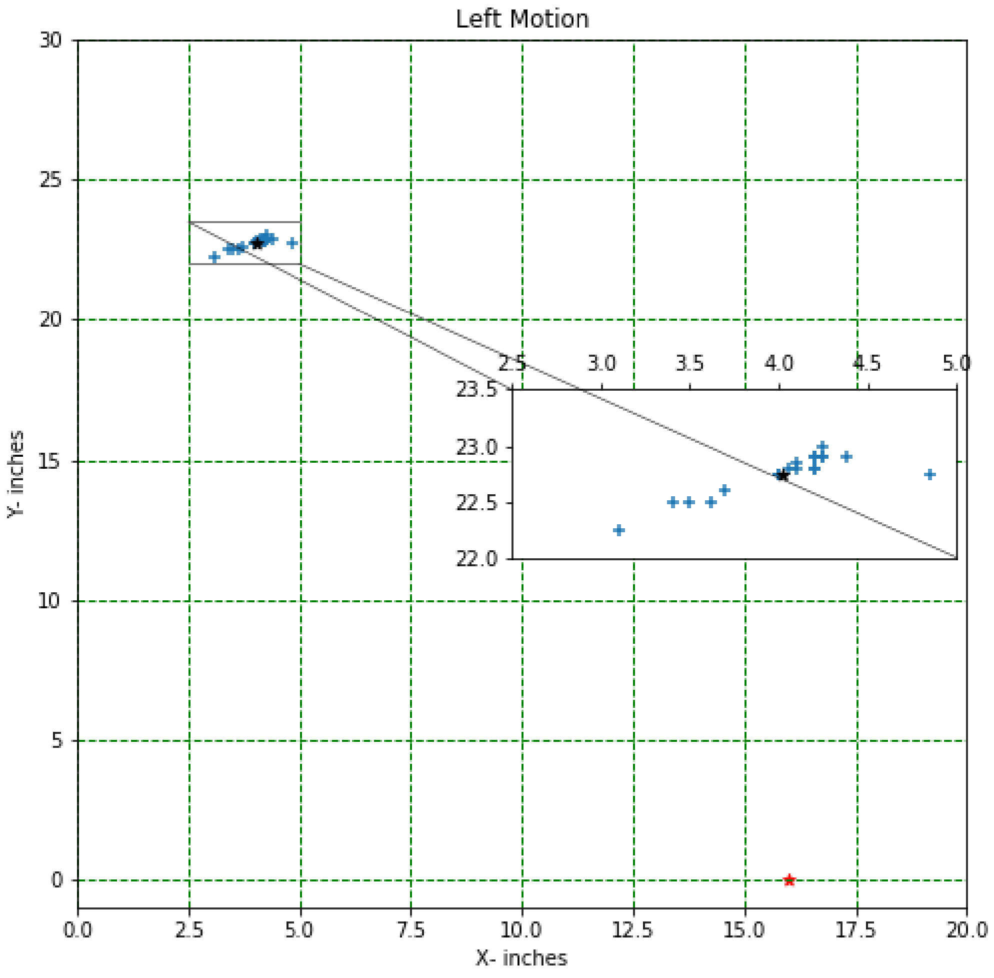


Figure 9: 20 iteration of moving left

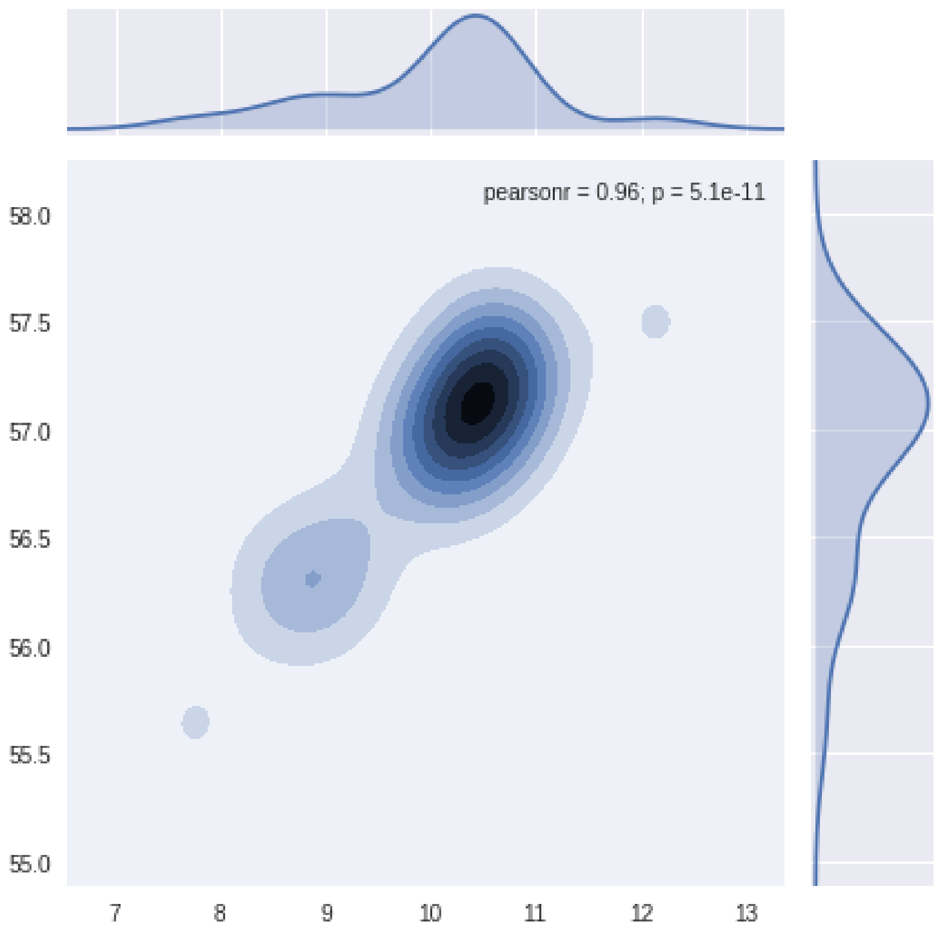


Figure 10: Density plot of (X, Y) for moving left

In the above figure it can be observed that the (X, Y) points obtained for left arc motion fit a gaussian distribution.

- $Range(x, y) = ([7.75 - 12.16], [55.63 - 57.50])$
- $Mean(x, y) = 10.05, 56.89$
- $Accuracy(x, y) = x \pm 0.5, y \pm 0.5$
- $\sigma(x, y) = 0.97, 0.47$
- $Mean(\theta) = 45.71 degree$
- $Accuracy(\theta) = \theta 7 \pm deg$
- $\sigma = 1.32 deg$
- $range(\theta) = 43.18 - 48.27$

**Boxplots**

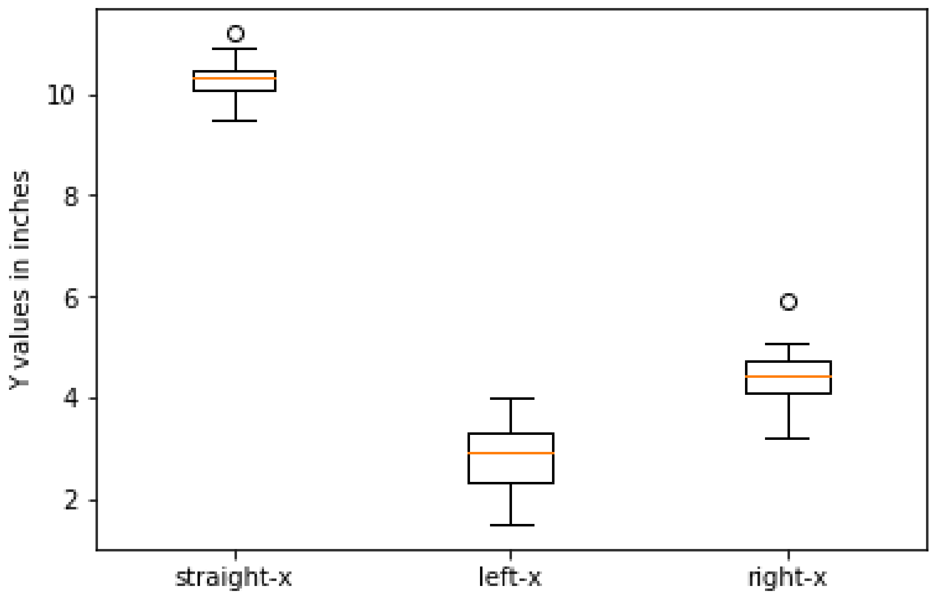


Figure 11: Box plot of X-values for straight line motion, left and right turn

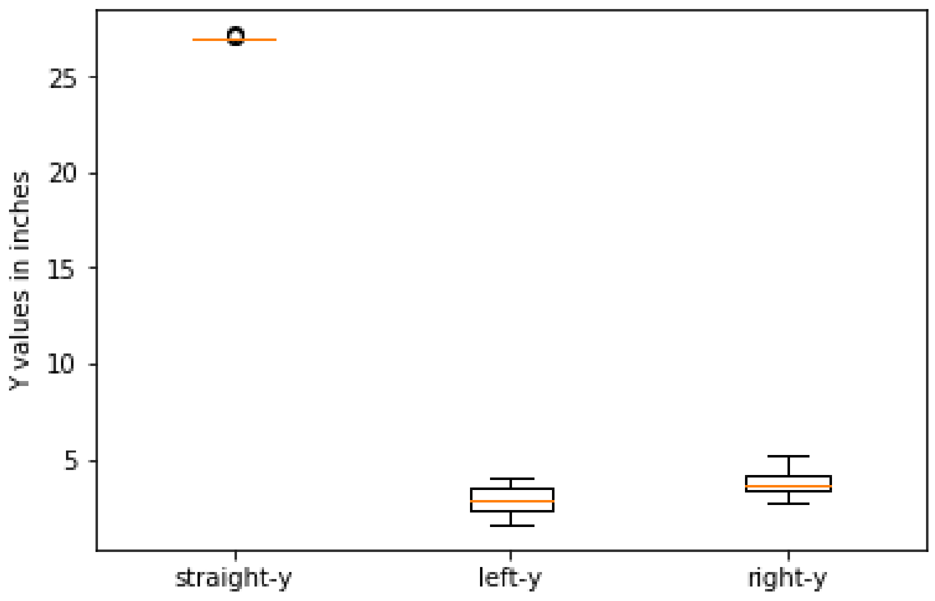


Figure 12: Box plot of Y-values for straight line motion, left and right turn

- 2.3 The chi-square test indicates that the observed data is a normal distribution.

```
In [9]: ks , p = stats.normaltest(straight_xy)
                alpha = 1e-3
        # null hypothesis: x comes from normal distribution
                if p.all() < alpha:
                        print("The null hypothesis can be rejected")
                else:
                        print("The null hypothesis cannot be rejected

[Out]: The null hypothesis cannot be rejected

In [10]: ks , p = stats.normaltest(right_xy)
                alpha = 1e-3
                if p.all() < alpha:
                        print("The null hypothesis can be rejected")
                else:
                        print("The null hypothesis cannot be rejected
[Out]: The null hypothesis cannot be rejected

In [11]: ks , p = stats.normaltest(left_xy)
                alpha = 1e-3
                if p.all() < alpha:
                        print("The null hypothesis can be rejected")
                else:
                        print("The null hypothesis cannot be rejected
[Out]: The null hypothesis cannot be rejected
```

- Deliverable 2.2: The functions used for plotting are from the seaborn library and use a kernel density method to obtain a 2-D density plot of the x and y coordinates.

- Deliverable 2.5:

## 1.6 Straight

  - In the case of straight line motion the readings obtained for Y coordinates appear to be both precise and accurate, whereas the X coordinates appear to be precise but not very accurate.

## 1.7 Right

  - The readings obtained for Y coordinates appear to be both highly precise and accurate, whereas the X coordinates appear to be precise but not very accurate.

## 1.8 Left

- The readings obtained for X coordinates appear to have low precision and accuracy, whereas the Y coordinates appear to be precise with a low accuracy.

# 2 Calibrating an Optical Tracking System

## 2.1 Setup for camera calibration

1. The camera calibration process is done on a checkerboard (calibration tool).
2. Installed the *MATLAB support package for USB webcam* for camera calibration.
3. The camera calibration toolbox for MATLAB is downloaded.
4. The webcam (Microsoft Lifecam) is connected to the laptop. The connectivity is checked by running *webcam list* command in MATLAB, which shows the list of all webcams connected to PC.
5. The webcam is mounted on a static paper punch, while capturing the images.
6. The Auto-focus is turned *off* by running the following command in MATLAB,
7. The images of checkerboard are captured from different perspectives (different position and orientation in the camera's field of view). A total of 20 images are captured and saved for the calibration process.

## 2.2 Estimation of number of images

- Since the object in the world is 3-dimensional and the captured images are 2D (co-planar), we loose one dimension. In order to regain the actual dimensionality during the calibration, various number of images are captured in different positions and orientations.
- As a rule of thumb, 20-40 image pairs is quite enough for the process. For our calibration we have captured 20 images in different position and orientations.
- The idea behind the capturing enough images is to cover the field of view, and to have a good distribution of 3D orientations of the board.

## 2.3 Description of the camera parameters

After the calibration, the list of parameters obtained are of two types,

1. Intrinsic parameters

- **Focal length:** Distance between the lens and the image sensor when the object is in focus, usually stated in millimeters. The focal length in pixels is stored in the 2x1 vector $fc$.
  *Calibration result:* $[648.5545, 648.4]$
- **Principal point:** A point at the intersection of the optical axis and the image plane. The principal point coordinates are stored in the 2x1 vector *cc*.
  *Calibration result:* 415.7172
- **Skew coefficient:** Defines the angle between x and y pixels axes and is stored in $alpha_c$
  *Calibration result:* 0
- **Radial Distortions:** Symmetric distortion caused by the lens due to imperfections in curvature when the lens was ground.
  *Calibration result:* $[-0.0187, 0.118]$
- The image distortion coefficients (radial and tangential distortions) are stored in the 5x1 vector *kc*.
  *Calibration result:* Tangential distortion $= [0, 0]$

2. Extrinsic parameters

- **Rotational Vectors:** Vector describing the rotation of camera plane with respect to the world frame.
  *Calibration result:* $3x3x20$ double.
- **Translational vectors:** Vector describing the translation of camera plane with respect to the world frame.
  *Calibration result:* $20x3$ double.

## 2.4 Possible problems or error sources that can disturb the calibration process

1. If the naming conventions of the images are different, then the calibration process gives error.

2. The computer running the calibration toolbox must have enough RAM(128 MB or less). If this condition is not met, then it gives *OUT OF MEMORY* error.

3. Enough number of images must be provided to get better calibration results.

4. The images captured with same orientations will not provide enough inputs for the toolbox to calibrate efficiently.

## 2.5 Image poses used for calibration

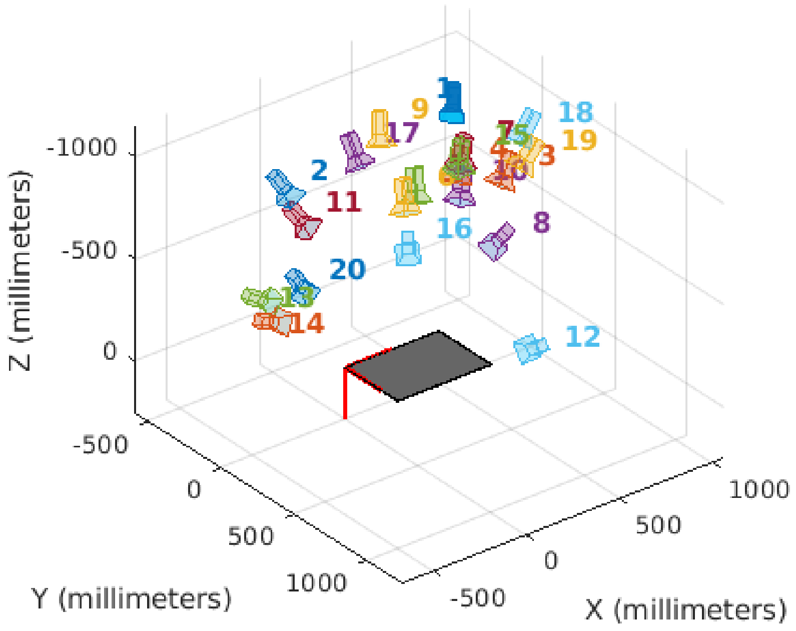


Figure 13: Pattern centric view

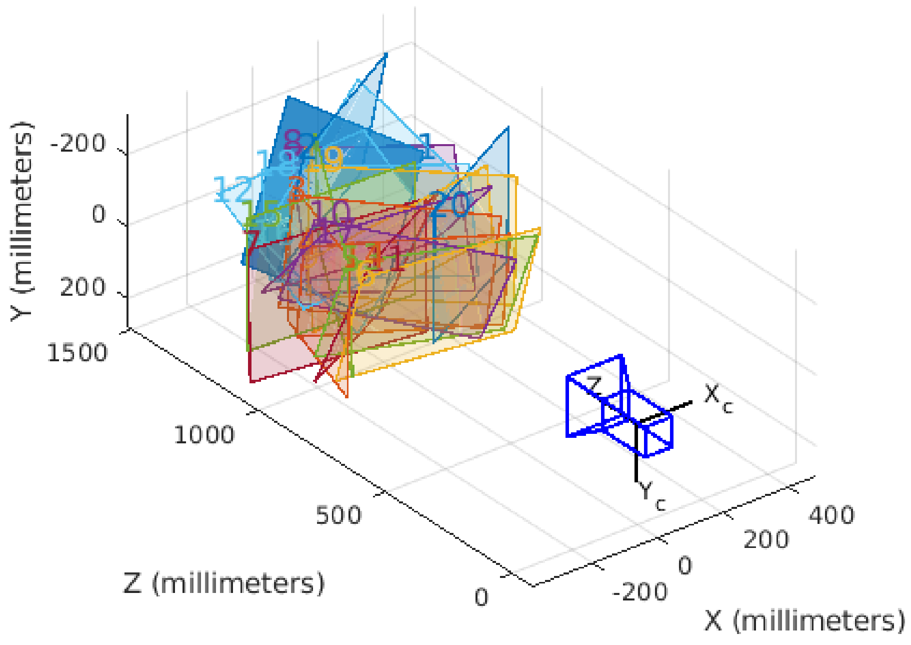


Figure 14: Camera centric view

## 2.6 Camera parameters obtained from calibration

$$IntrinsicMatrix = \begin{bmatrix} 648.7381 & 0 & 0 \\ -0.5603 & 648.5722 & 0 \\ 417.6238 & 285.6146 & 1 \end{bmatrix}$$

$$FocalLength = \begin{bmatrix} 648.7381 & 648.5722 \end{bmatrix}$$

$$Principalpoint = \begin{bmatrix} 417.6238 & 285.6146 \end{bmatrix}$$

$$Skew = -0.5603$$

$$Radialdistortion = \begin{bmatrix} -0.0183 & 0.1165 \end{bmatrix}$$

$$TangentialDistortion = \begin{bmatrix} -0.0016 & 9.2118e-04 \end{bmatrix}$$

$$Meanprojectionerror = 0.1516$$

## 2.7 Estimation errors

### 2.7.1 Intrinsic errors

$$Skewerror = 0.1173$$

$$Focallengtherror = \begin{bmatrix} 0.3475 & 0.3245 \end{bmatrix}$$

$$Principalpointerror = \begin{bmatrix} 0.6292 & 0.5130 \end{bmatrix}$$

$$Radialdistortionerror = \begin{bmatrix} 0.0028 & 0.0134 \end{bmatrix}$$

$$Tangentialdistortionerror \begin{bmatrix} 2.8109e-04 & 3.5275e-04 \end{bmatrix}$$

## 2.8 Arguments for which camera model best fits

If the lens distortions are really too severe (for fish-eye lenses for example), the simple guiding tool based on a single distortion coefficient kc may not be sufficient to provide good enough initial guesses for the corner locations. then corner extraction has to be done manually.

# 3 Appendix

## 3.1 Softwares used

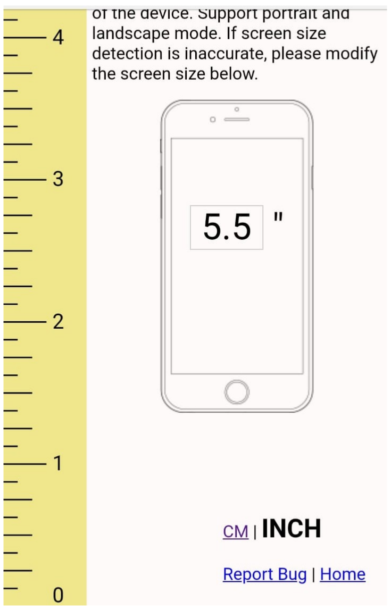
- Python 2.7
- Jupyter Notebook

## 3.2 Packages used

- Matplotlib
- Numpy
- Scipy
- Seaborn
- Pylab

## 3.3 Tools used

- Onscreen ruler is used to mark scale on the sheet (metrics in inches)
- `https://www.piliapp.com/actual-size/inch-ruler/`


of the device. Support portrait and landscape mode. If screen size detection is inaccurate, please modify the screen size below.

5.5 "

Figure 15: Online ruler

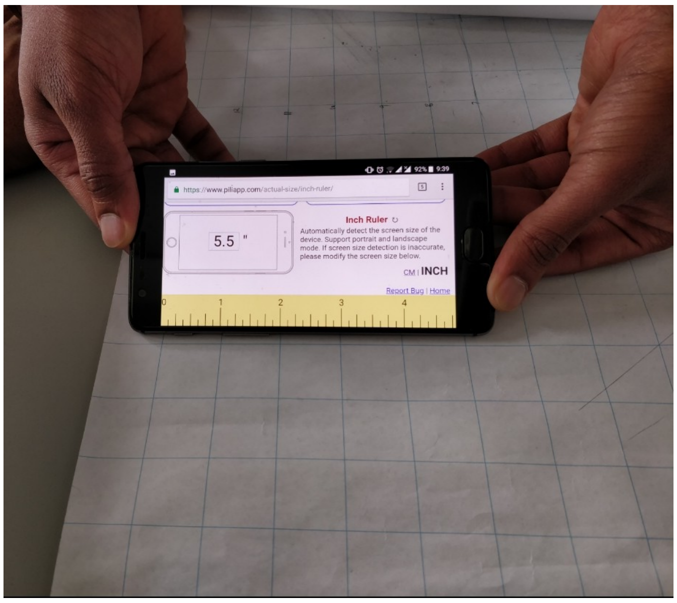

https://www.pillapp.com/actual-size/inch-ruler/
5.5 "
Inch Ruler
Automatically detect the screen size of the device. Support portrait and landscape mode. If screen size detection is inaccurate, please modify the screen size below.
CM | INCH
Report Bug | Home
0
1
2
3
4

Figure 16: Marking scale on the sheet using onscreen ruler