# SYMMETRIC POINT CLOUD

Freeform Surfaces Project Report


Srini Prakash Maiya (923123)

Saad Masood (998190)

Guide:

Prof. Dr.-Ing. Markus Lazar

# Acknowledgement

This report was generated at the Technical University of Applied Sciences, Rosenheim in the Winter Semester of 2021 (Sep 2021 –Feb 2022)

At this point we want to wholeheartedly thank our Academic and Project guide, Prof. Dr.-Ing. Markus Lazar for providing us with the opportunity to work on this project. We are also thankful for his guidance and technical support.

# Declaration

Hereby us, Srini Prakash Maiya and Saad Masood declare corresponding to § 35 passage 7 of Rahmenprüfungsordnung for advanced technical colleges in Bavaria, that the project report with the title

## Symmetric Point Cloud

Is compiled by ourselves and is not exhibited for other examination purposes. Equally, all the resources used are referenced and have not used anything undeclared.

# Contents

# TABLE OF FIGURES

# 1. Introduction

## 1.1. Point Cloud

A point cloud is essentially a large collection of individual 3D points. It is one of the most primitive form for a 3D representation. The points can additionally also include RGB color data or even intensity information of the strength of the laser pulse that generated the point.

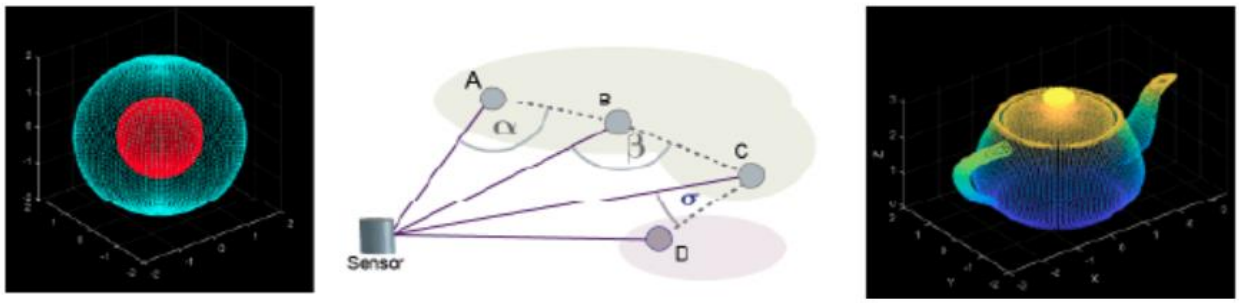



*Figure 1 Point Cloud Visualization [1]*

The generated point clouds can be further converted into polygon or triangle meshed to generate a 3D CAD model for ease of visualization and further modification.

- Generation of Point Clouds:

  There are two widely used tools used to capture a point cloud. Namely,

1. Laser scanners: Is a survey-grade system widely used in commercial and industrial applications. This scanner includes different sensors and technologies. LiDAR (Light Detection and Ranging) is the most important sensor, which emits rapid pulses or laser to gather a thousands of accurate measurements per second.

   The scanner processes the received laser beam pulse to combine the vertical and horizontal angles to calculate a 3D X, Y, Z coordinate position for each point to produce a set of 3D coordinate. Most laser scanners also include an RGB camera to add color to the point cloud and an inertial measurement unit (IMU), a much accurate accelerometer.

   The widespread usage of laser scanners in industrial applications can be attributed to their higher degree of sophistication when compared to alternative point cloud creation methods.

2. Photogrammetry: Is a method to extract 3D information from a scene using multiple photographs. In this method, multiple overlapping photographs of the object is captured. The photos are overlaid on one another and algorithms are used to form continuity between a photo and it's next, thus making it possible to extract 3D data from multiple 2D images.

This method cannot be used in real time point cloud generation as the further processing of a large number of photos take a considerable amount of time. This method also requires a constant illumination to take the photos. [2]

## 1.2. Symmetry

If a form can be broken into two more identical parts and positioned in an orderly manner, it is said to be symmetric.

- Types of symmetry:

1. Rotational symmetry / Radial symmetry: The property of a shape which makes it look the same after a partial rotation has been applied on it. The degree of rotational symmetry is the number of distinct orientations a shape can be rotated without changing its appearance. [3]

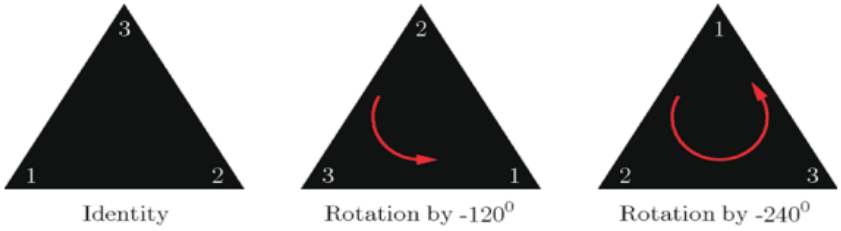



*Figure 2 Rotational symmetry of an equilateral triangle [3]*

2. Translational symmetry: When moved from point $A \rightarrow A + t$ with no rotation, if the body maintains its original shape, then the body is said to have translational symmetry. Implies that at least in one direction the object is infinite. [4]

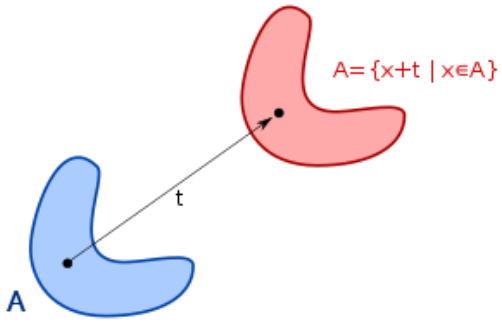



*Figure 3 Translational symmetry [4]*

3. Reflection symmetry/ Mirror symmetry: If the body does not undergo change upon undergoing a reflection along its axis of symmetry, then the figure is said to have reflectional symmetry.

   A 2D figure contains line/ axis of symmetry. In a 3D body the plane is used to define the reflection symmetry. A body can possess multiple axes or planes of symmetry. [5]

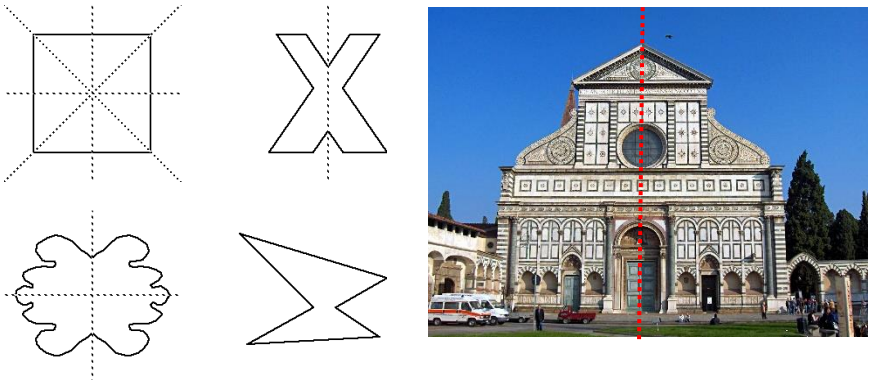


*Figure 4 Reflection symmetry [5]*

4. Bilateral Symmetry: The objects possessing this type of symmetry contain only one plane of reflection symmetry, i.e. along the sagittal plane. This plane of symmetry divides the bodies into exact two halves.

   Most of the animals, insects including humans are roughly bilaterally symmetric.


*Figure 5 Bilateral symmetry [14]*

# 2. Methodology

This project aims to familiarize with point cloud processing, available point cloud processing methods, softwares and to understand the mathematics behind processing of the point clouds.

## 2.1. Given Task

This project was divided into two sections. In the first section, the bilateral symmetry plane of the complete point cloud was to be determined. In the second section, the unknown points of the incomplete point cloud were to be interpolated with the help of the information of bilateral symmetry of the object.

- Subdivided tasks:

### *2.1.1. Task 1: To find symmetry plane of the complete point cloud.*

- Input: An arbitrarily oriented three-dimensional point cloud with 10,000 points. The point cloud is produced as a result of a laser scan of an object. The scanned object possesses a bilateral symmetry.
- Aim: To find the bilateral symmetry plane of the given point cloud.

### *2.1.2. Task 2: To interpolate the missing points of the complete point cloud*

- Input: An arbitrarily oriented three-dimensional point cloud with 8372 points. This point cloud resembles the generated point cloud due to the result of an incomplete scan.
- Aim: To interpolate the missing points using the information of bilateral symmetry of the point cloud.

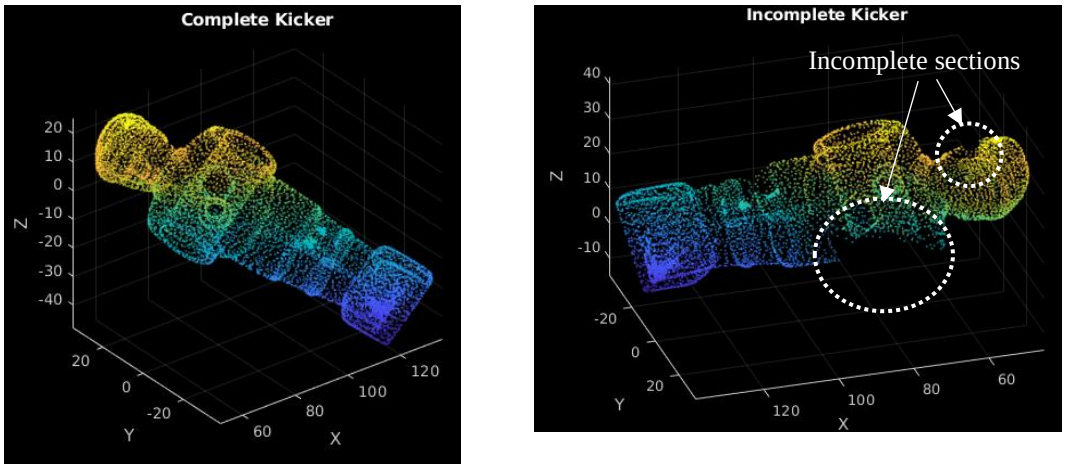



*Figure 6 Arbitrarily oriented complete (left) and incomplete (right) point clouds before processing*

## 2.2. Software

The whole project was carried out using MATLAB. It has an extensive inbuilt library dedicated to point cloud visualization, processing and registration processes. It has also a detailed documentation for each function/ command. Additional Toolboxes can be installed depending on the type of processing of point cloud, ranging from general visualization to LiDAR-SLAM mapping and point cloud segmentation using Deep neural networks.

Version of MATLAB: 2021a

The inbuilt 'Point Cloud Processing' library was used for all the point cloud processing and visualization purposes. The important inbuilt functions used in this project were:

- pointCloud: Object for storing 3-D point cloud. Converts the given points into 'pointCloud' format for further processing. [6]
- pcshow: Plot a 3-D point cloud.
- pcshowpair: Visualize difference between two point clouds
- pcdenoise: Removes noise from 3-D point cloud.
- pcdownsample: Downsample a point cloud
- pcregistericp: Register two point clouds using ICP algorithm

A detailed documentation for all the above functions and the entire Point Cloud Processing library can be found at https://de.mathworks.com/help/vision/point-cloud-processing.html

# 3. Task 1: Bilateral symmetry plane detection of complete point cloud

In this task the complete point cloud is analyzed to extract the bilateral symmetry plane from the object. The object is aligned to the origin of the coordinate systems using a custom function. The object is aligned such that the center of the point cloud is at the origin of the coordinate system and the highest eigenvector aligned to the 'Z' axis. The second highest eigen vector of point cloud can be aligned either to 'X' or 'Y' axis depending on user input.

## 3.1. Procedure:

### *3.1.1 Reading the data*:

The given data is an Excel file (.xlsx). Such files can be read directly into MATLAB using the function *'readtable'* [7]. This function can read a variety of file types such as text(.txt), Excel (.xlsx, .csv) and HTML.

The output of the file is a table containing the data and this data is converted to an array using *'{}'* operator as shown in the image below. The read arbitrarily-oriented point cloud is visualized using the command *'pcashow'*.

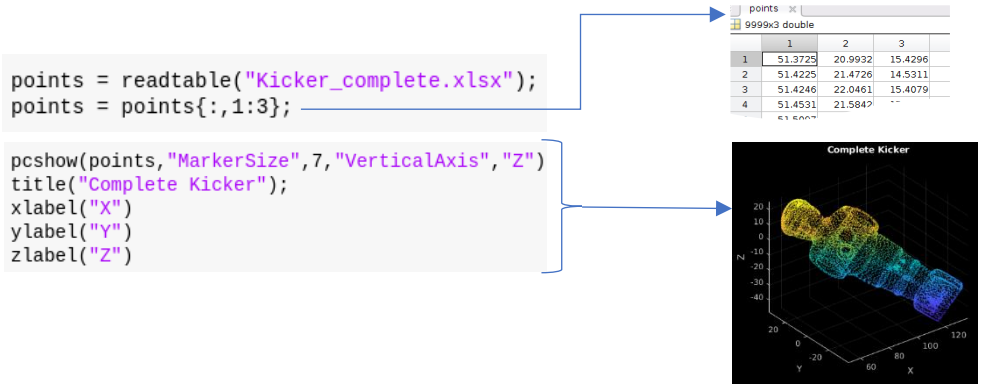



*Figure 7Reading and visualization of the point cloud*

### *3.1.2 Aligning the point cloud:*

A custom function from MATLAB File Exchange, *'rotatePointCloudAlongZ'* [8] was used to align the whole point cloud from its arbitrary location to the center of the coordinate system. The highest eigen vector of the point cloud is aligned to the 'Z' direction and with an user input the second highest eigenvector can be aligned either along 'X' or 'Y' direction of the coordinate system.

By aligning the point cloud to the center of the coordinate system, the point cloud can be better visualized.

This function shifts the center of the point cloud by subtracting mean/ center of gravity of the point cloud from each points of the point cloud. Principal Component Analysis (PCA) is performed on this centered point cloud. The eigenvectors originated from the PCA are converted into angles of

the vector along x- and z- axis by taking the arctangent of the vector. The whole point cloud is rotated by the resulting angles. The highest eigen vector is aligned to z-axis.

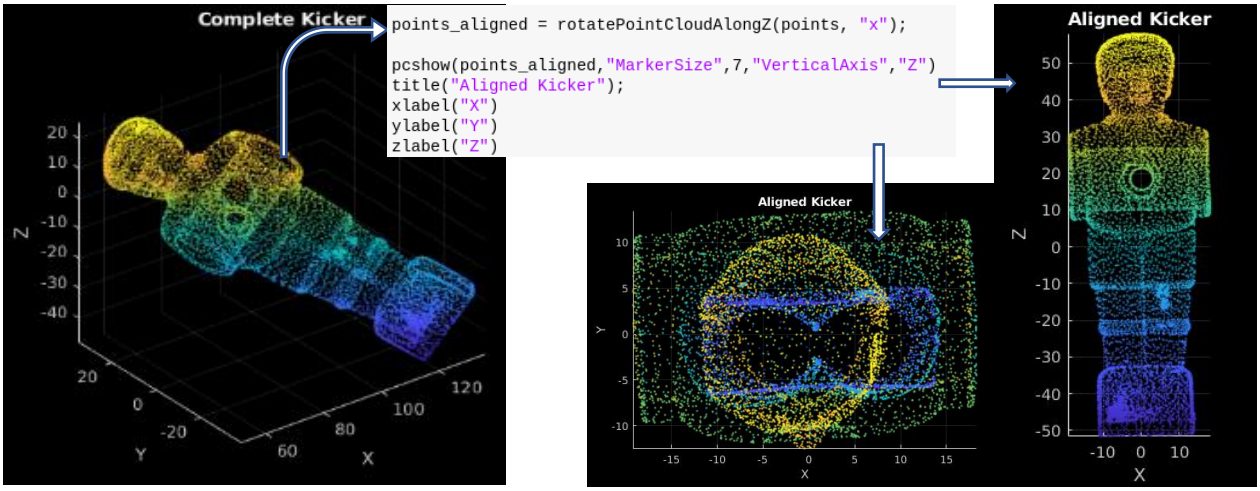



*Figure 8 Aligning point cloud to the origin*

### *3.1.3 Construction of bilateral symmetry plane:*

As the point cloud is already oriented and centered, the symmetry plane was drawn by taking the maximum and minimum values of the points along the y- and z- axes.

The center of the point cloud is located at the origin of the coordinate system and is indicated in the figure below.

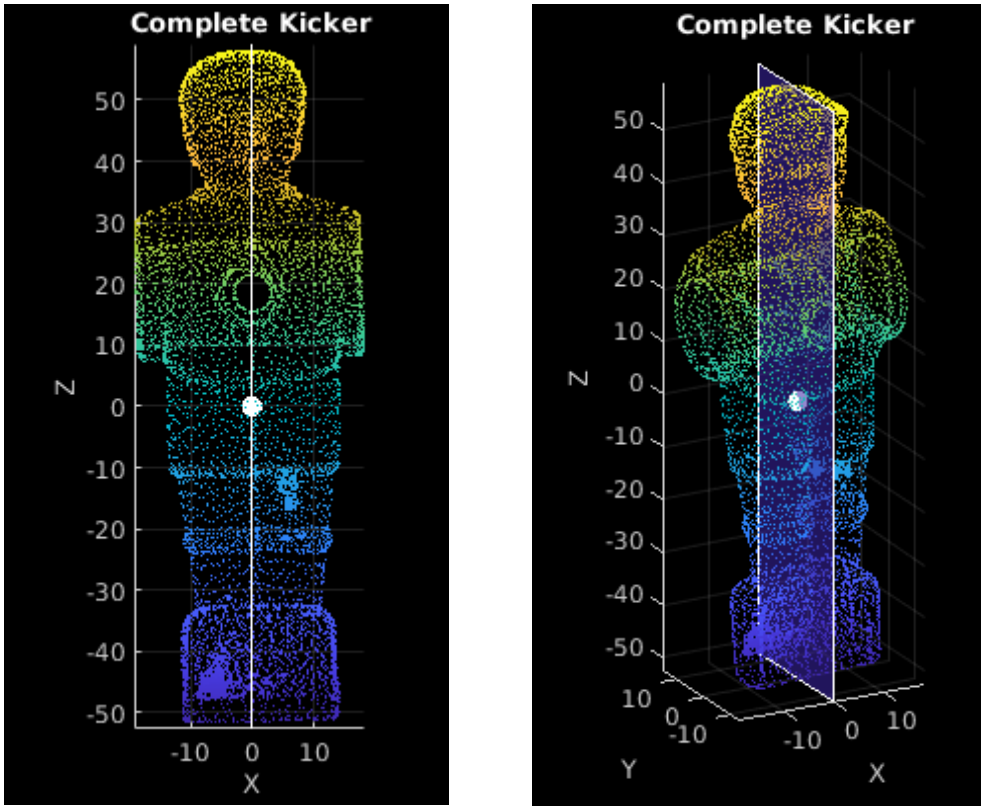



*Figure 9 Bilateral symmetry plane of the point cloud*

## 3.2 Further refinement:

The results are thoroughly analyzed and further refinements are carried out to yield a better result.

Problem and causation: Due to the inhomogeneous distribution of the points, i.e. the dense accumulation of the points in some parts of the point cloud causes slight tilt in the orientation of the figure. This results in a slight offset of the symmetry plane. The uneven symmetry plane is clearly visible in front view of the point cloud.

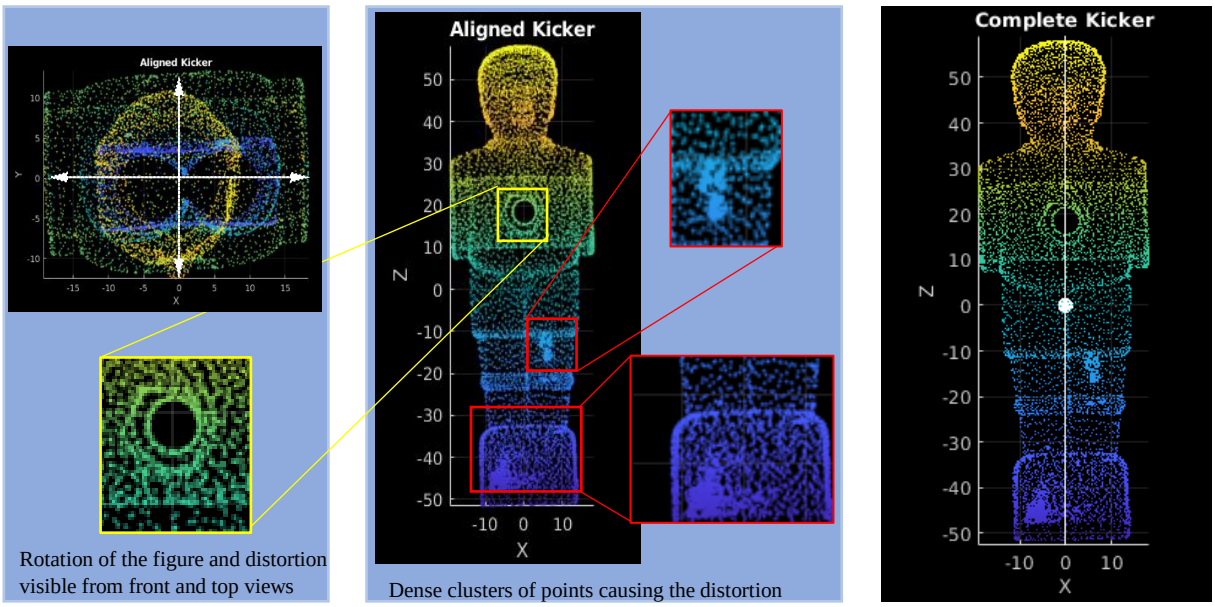
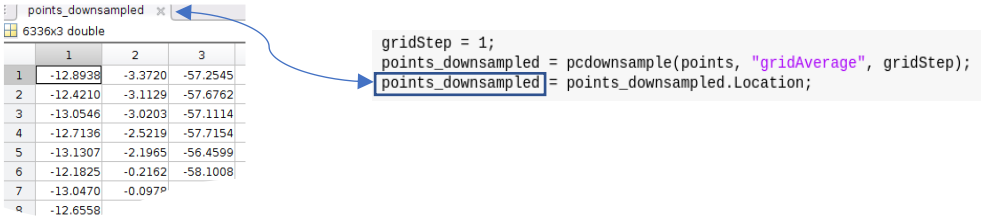



*Figure 10 Rotational distortions (left) caused by the dense point clusters (center) and an uneven symmetry plane (right)*

Solution:

- The to-be-processed point cloud is downsampled by inbuilt function, *'pcdownsample'* [9] . As this function takes a *'pointCloud'* object as input, the list of points is converted to a point cloud object and downsampled. The 'gridAverage' value is set to 1, indicating all multiple points in a grid of 1x1x1 are merged into a single point.




- The downsampled point cloud object is converted back to array format and is aligned again to the origin of the coordinate system. The point cloud is reduced by 3664 points, resulting in a 3-dimensional array of 6336 rows.
- The point cloud is aligned again to the origin and the bilateral symmetry plane is extracted again.

- In the below figure a side-by-side comparison between resulting symmetry plane of the original point cloud and the downsampled point cloud is made. Prominent areas of deviation and improvement are highlighted.

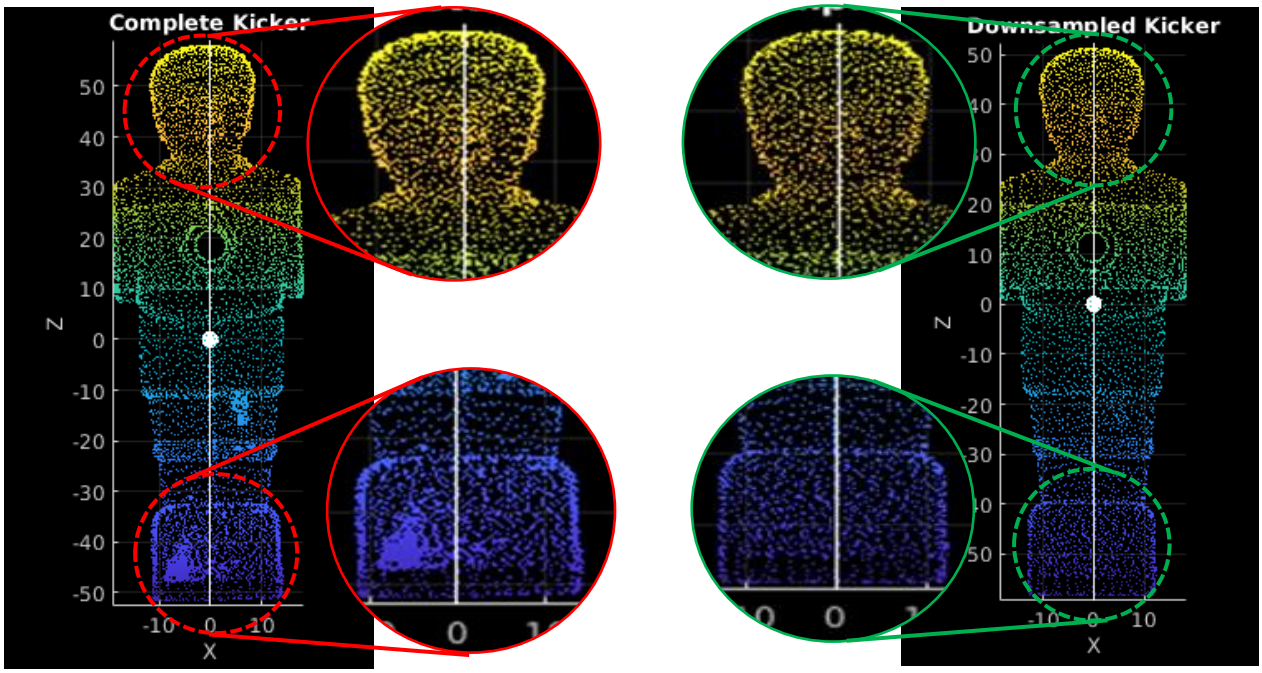
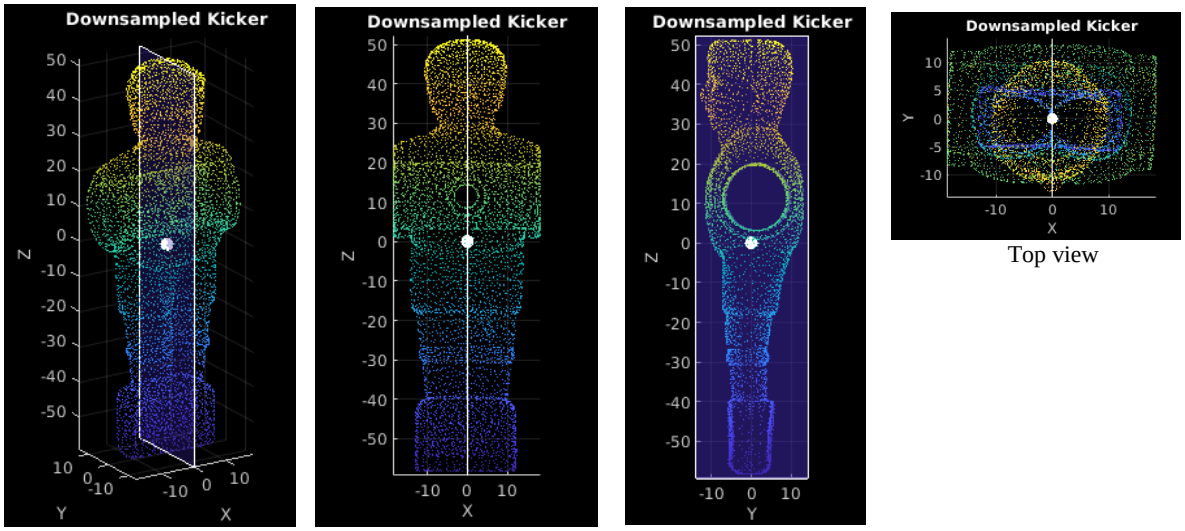



*Figure 11 Comparison of extracted bilateral symmetry plane of original point cloud (left) and donwsampled point cloud (right)*

The output of the downsampled and oriented point cloud is written to a csv file and saved. This file is used as a reference point cloud for the task 4.1 to interpolate missing points using the inbuilt functions of MATLAB.

The final result of the obtained bilateral symmetry plane is shown in the figure below.




Isometric view | Front view | Left-side view | Top view

*Figure 12 Multiple views of the obtained bilateral symmetry plane*

# 4. Task 2: Interpolation of missing points of an incomplete point cloud

In this task, the input is an incomplete point cloud with 8732 points. The missing points were interpolated by two methods, namely

## 4.1 Interpolation by inbuilt functions (pcregistericp [10])

### *4.1.1. Overview*

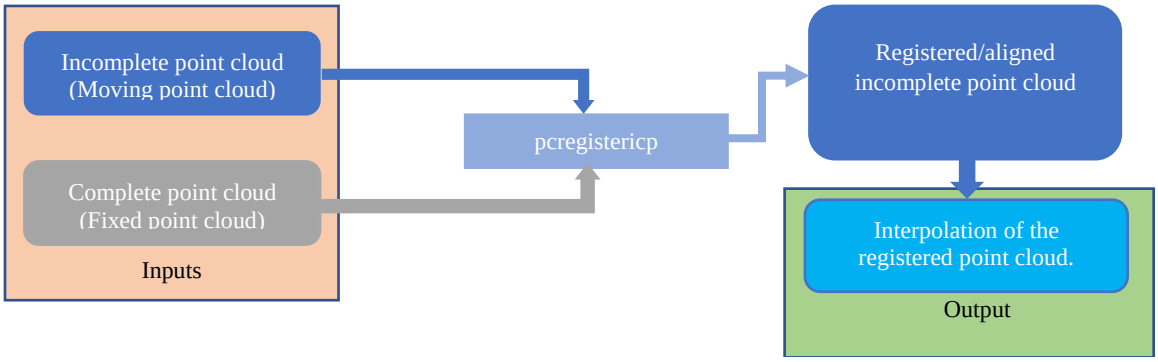



*Figure 13 Workflow of interpolation using 'pcregistericp' algorithm*

In this approach of interpolation, MATLAB's inbuilt point cloud registration algorithm is used. Registration algorithms are used to associate the two or multiple sets of data points into a common coordinate system. These algorithms are used in object reconstruction, inspection and localization of objects in which the point clouds have to be compared or registered [11].

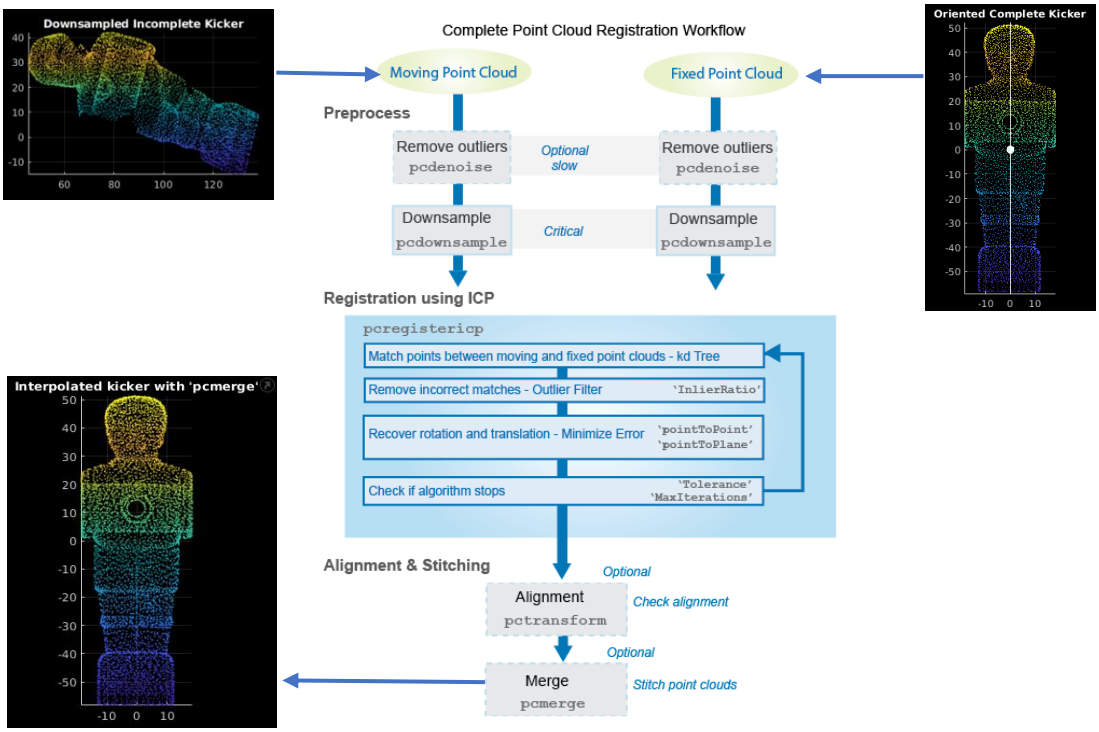



*Figure 14 Overview of the workflow of registering an incomplete point cloud with respect to fixed point cloud*

- *'pcregistericp'* algorithm:
  This algorithm is based on iterative closest point (ICP) algorithm. Using a kd-tree [12], an algorithm that finds nearest point to the given point in a 3D space of points, the '*pcregistericp'* algorithm of MATLAB finds the most appropriate point of the incomplete point cloud with respect to the complete point cloud. Thus, we obtain the incomplete (Moving) point cloud oriented in the same orientation of the complete (Fixed) point cloud.

### *4.1.2 Workflow*

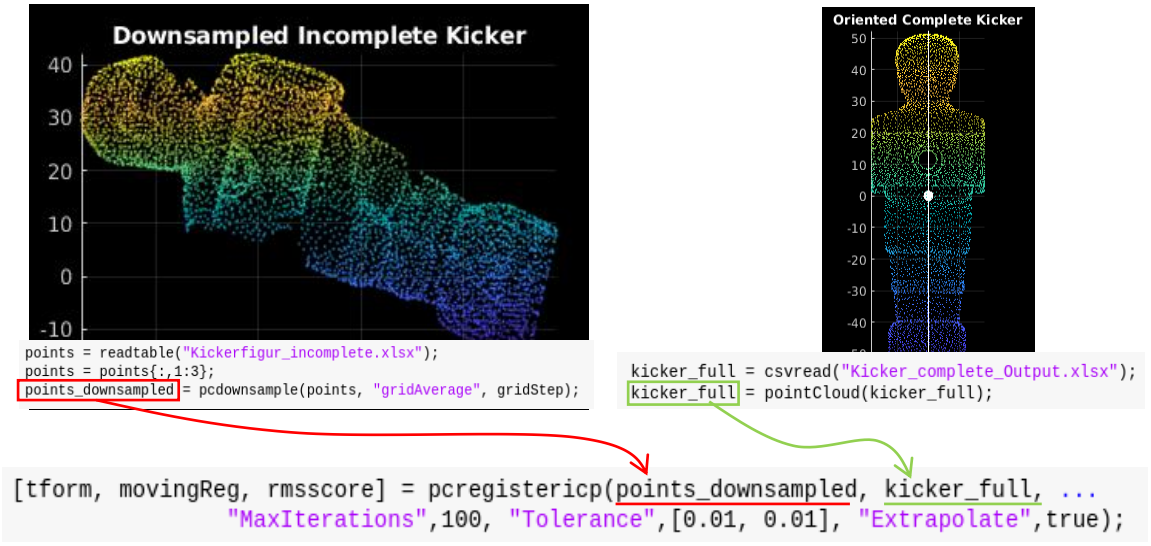



*Figure 15 Procedure of ICP registeration using MATLAB function*

- The incomplete point cloud is read and converted as a point cloud object.
- The saved point cloud from Task 1 is used as a reference point cloud and the read csv file is converted to a point cloud object.
- These two point cloud objects are passed to the algorithm along. The additional parameters of 100 iterations, with a tight translational and rotational tolerance of 0.01 each and the additional extrapolation step to trace out the registration is provided by setting the parameter 'Extrapolation' to true.

### *4.1.3 Results*

The output of this algorithm consists of a rigid transformation matrix *'tform'*, a registered point cloud *'movingReg'* and a root mean square(rms) value indicating the error rate of registration *'rmsscore'*.

The point cloud was moved -37.83 units in x-, -55.28 units in y- and 62.56 units in z-axes as indicated by the figure below.

A rms value of 0.217 was obtained indicating a small amount of error, thus assuring a good registration.

The rotation matrix was extracted from the rigid transformation matrix *'tform'* and was further converted into angles to analyze the rotation caused by the registration.

```
tform.Translation

ans = 1×3
   -37.8332   -55.2840   62.5596
```

Translation along: x y z

```
tform.Rotation

ans = 3×3
    0.2433    0.6068   -0.7567
   -0.2394    0.7936    0.5594
    0.9399    0.0451    0.3384
```

```
rmsscore

rmsscore = 0.2174
```

*Figure 16 Translation (left), Rotation (center) and Root Mean Squared Error value of registered point cloud*

The rotation matrix is further converted into angles, to analyze rotation caused due to the registration in each axis.

```
tform.Rotation

ans = 3×3
    0.2433    0.6068   -0.7567
   -0.2394    0.7936    0.5594
    0.9399    0.0451    0.3384
```

```
rotMat = tform.Rotation;
theta_x = atan2(rotMat(2,2), rotMat(3,3))*180 / pi

theta_x = 66.9077

theta_y = atan2(-rotMat(3,1), sqrt(rotMat(3,2)^2+ rotMat(3,3)^2))*180 / pi

theta_y = -70.0409

theta_z = atan2(rotMat(2,1), rotMat(1,1))*180 / pi

theta_z = -44.5370
```

*Figure 17 Conversion of rotation matrix to Euler angles in X, Y and Z axes*

It can be observed that incomplete point cloud is 66.9° in x-, -70.04° in y- and -44.54° in z- axes to align with the reference point cloud.

The resulting registered point cloud is still incomplete but it is aligned to the complete point cloud as shown in the below figure.

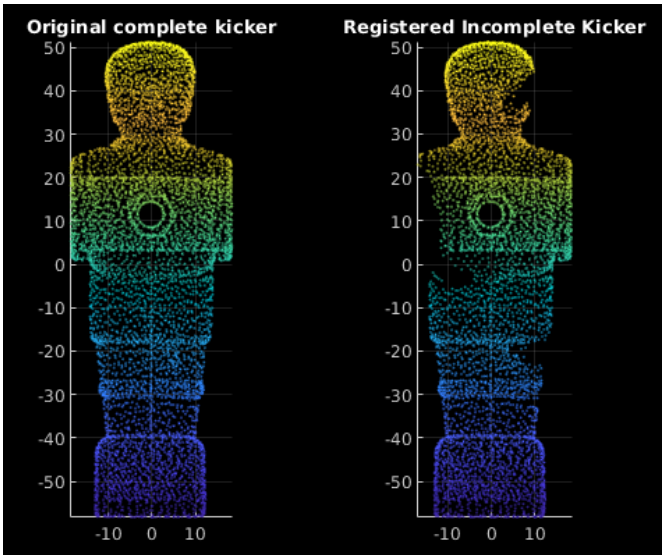



*Figure 18 Side-by-side view of original/reference point cloud to the registered point cloud*

### *4.1.4 Interpolation of points*

The points were interpolated using two methods.

- *'pcmerge'* function of MATLAB
- Manual interpolation of points

### *4.1.4.1 Merging point clouds by 'pcmerge' [13]*

The inbuilt MATLAB command *'pcmerge'* was used to directly merge the registered point cloud with the reference point cloud to interpolate the missing points.

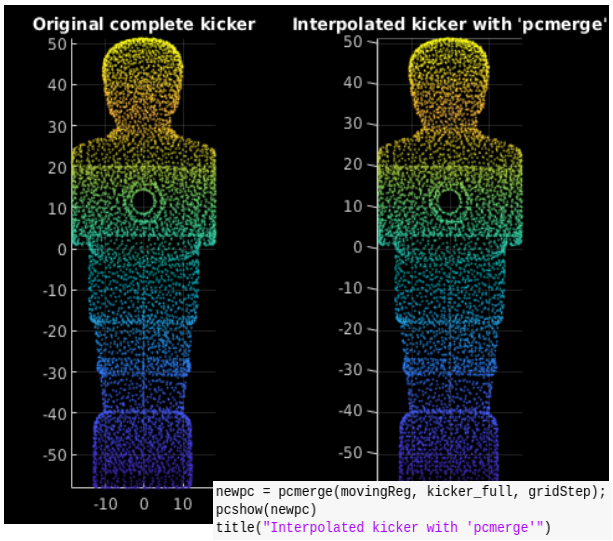



*Figure 19 Interpolation of points by 'pcmerge' (right)*

### *4.1.4.2 Interpolation of points*

Here every point of registered point cloud is mirrored along the bilateral symmetry plane to interpolate the missing points of the point cloud. As the figure is oriented at its exact center due to the registration process, we get a point cloud double the size of the original, but all the points mirrored along the bilateral symmetry plane.

```
x = 1;
for i = 1:num_rows
    point = movingReg(i,:);
    point(1) = -1*point(1);

    movingReg = [movingReg
                 point ];
    if mod(i,300) == 0
        figure(x)
        pcshow(movingReg, "MarkerSize",10)
        hold on
        x = x+1;
        view([0 0])
        i
    end
end
```

For every row

- Take the point of the correcponding row — 1
- Multiply the x coordinate value of the point by -1

Append the flipped point to point cloud — 2

Visualize the plots at interval of 300 iterations — 3

*Figure 20 Interpolation of points (left) and pseudo code (right)*

The visualization of interpolation of points is shown in the figure below at different critical values of the rows. The interpolated points can be visualized by the dense regions of the point cloud.

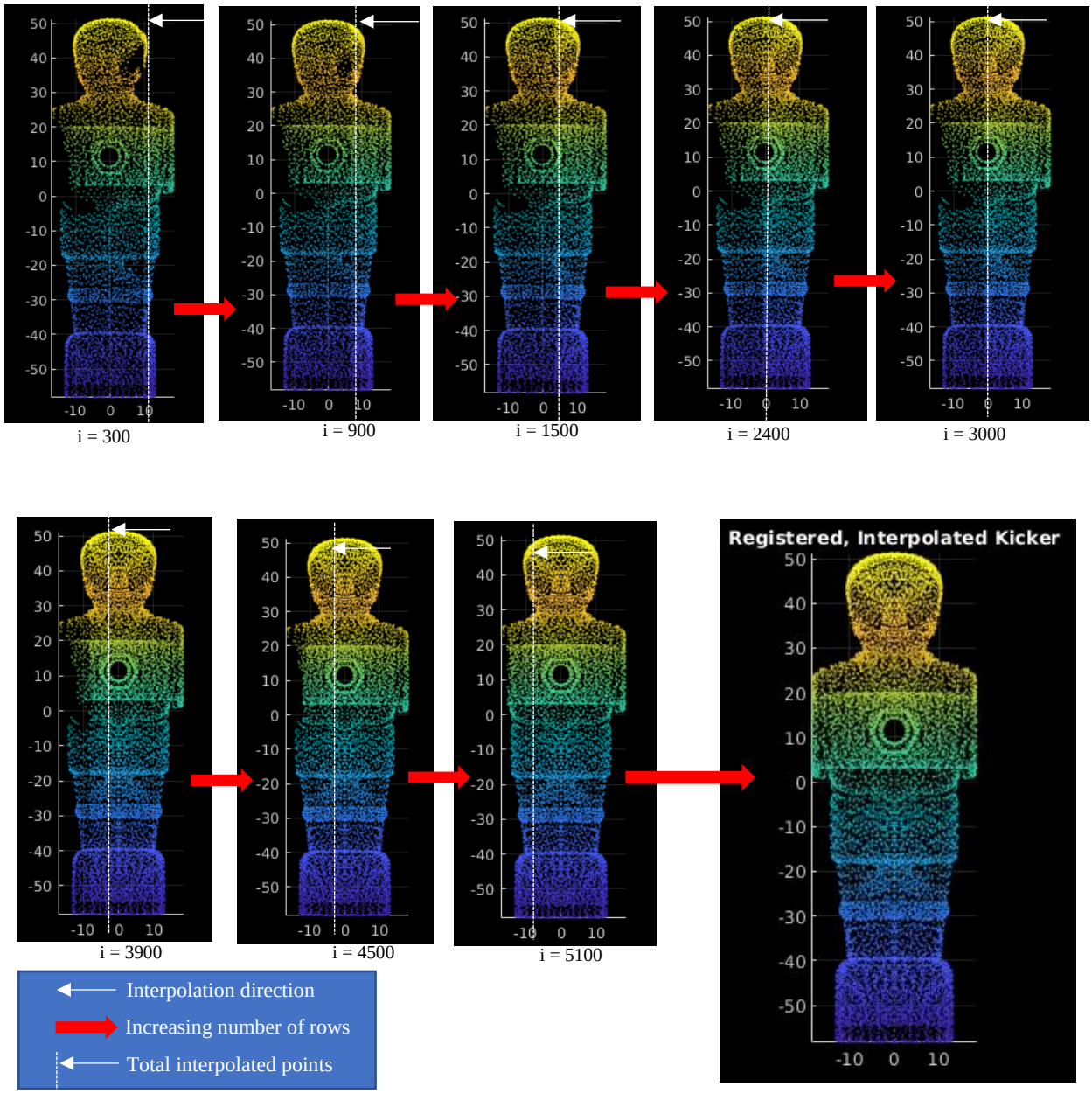



*Figure 21 Visualization of manual interpolation of missing points*

## 4.2 Manual interpolation of points with no registration algorithms

Although the registration functions of MATLAB provide a high level of accuracy, sophistication and robustness, to find the missing points of the incomplete point clouds a complete reference point cloud is a must.

In this section, an interpolation using pure rotational matrices has been tried out which do not require any interpolation methods and a reference point cloud.

The point cloud is oriented to its approximate center of gravity and the approximate bilateral symmetry plane is extracted. On this oriented point cloud, a different combination of rotations and displacements are applied. For each combination of translation and rotation of the point cloud, a root mean square metric is stored along with the rotation angles and translation value.

The root mean square metric calculates the sum of Euclidean distances of all points from the approximate center. The minimum value of this metric implies at that particular orientation of the point cloud, the point spread is minimized. The incomplete point cloud is set to this orientation and is interpolated to get a good fit of interpolated points.

### *4.2.1 Procedure*

#### *4.2.1.1 Approximate alignment of the point cloud*

The incomplete point cloud is oriented from its arbitrary position to the center of the coordinate system with the help of *'rotatePointCloudAlongZ'* function by using the similar workflow as before. Due to the incompleteness of the point cloud, the calculated center and the bilateral symmetry planes are offset and are not at a nominal value.

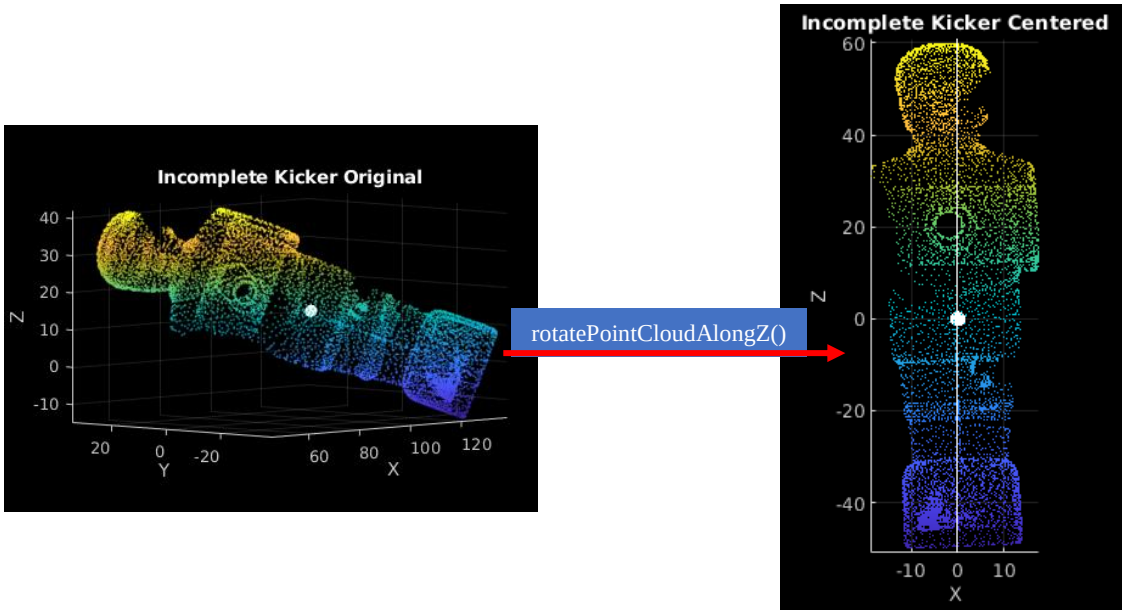



*Figure 22 Extraction of approximate symmetry plane from an incomplete point cloud*

This approximate plane of symmetry is taken as reference several combinations of rotation and translations are performed. The point cloud is downsampled using *'pcdownsample'* function to remove the dense clusters of points as shown in the below figure.

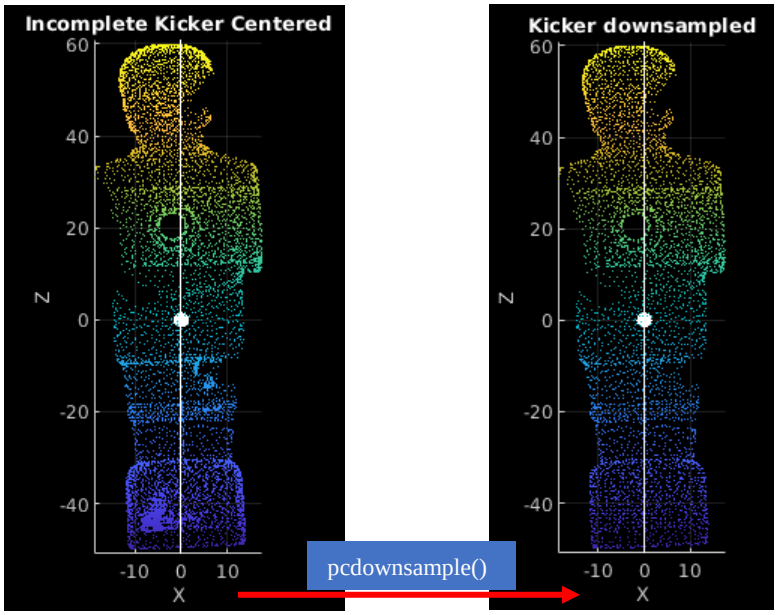



*Figure 23 Downsampling of the aligned point cloud*

### *4.2.1.2 Direct interpolation*

A direct interpolation of the downsampled point cloud is done using the same methodology as described in section 4.1.4.2. The results are visualized below.

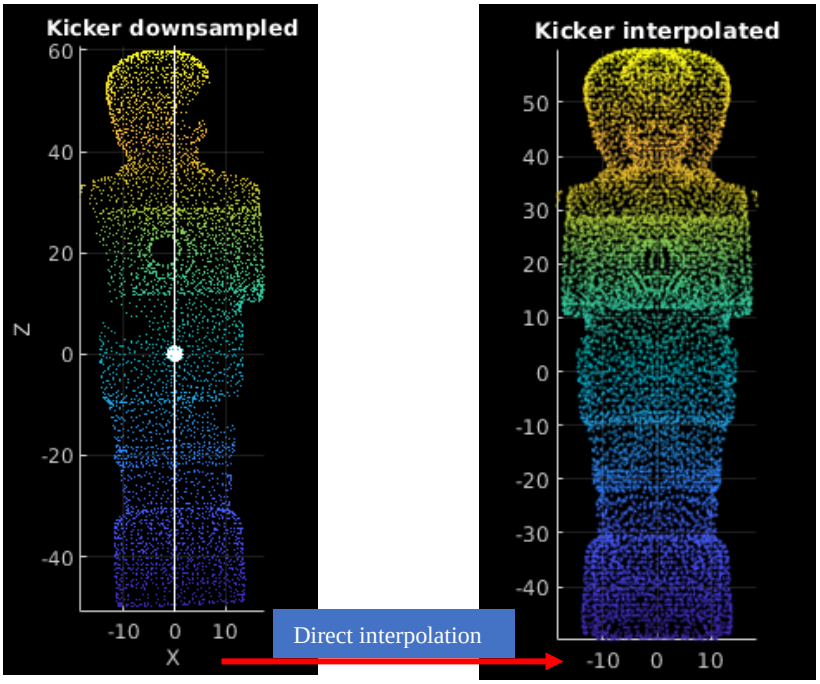



*Figure 24 Result of direct interpolation of the point cloud using the approximate symmetry plane*

As from the above figure, it is clearly evident that the interpolated point cloud has severe distortion due to the incorrect orientation of the point cloud and symmetry plane. Further refinements are done to reduce the distortion of the point cloud interpolation.

### *4.2.1.3 Further refinements*

Problem and causations:

- It was observed that the center of the aligned incomplete point cloud was shifted along the negative z axis when it was visually compared with the complete point cloud.
- The rotation of the point cloud in the z- and y- axes were prominent when compared to the x-axis.

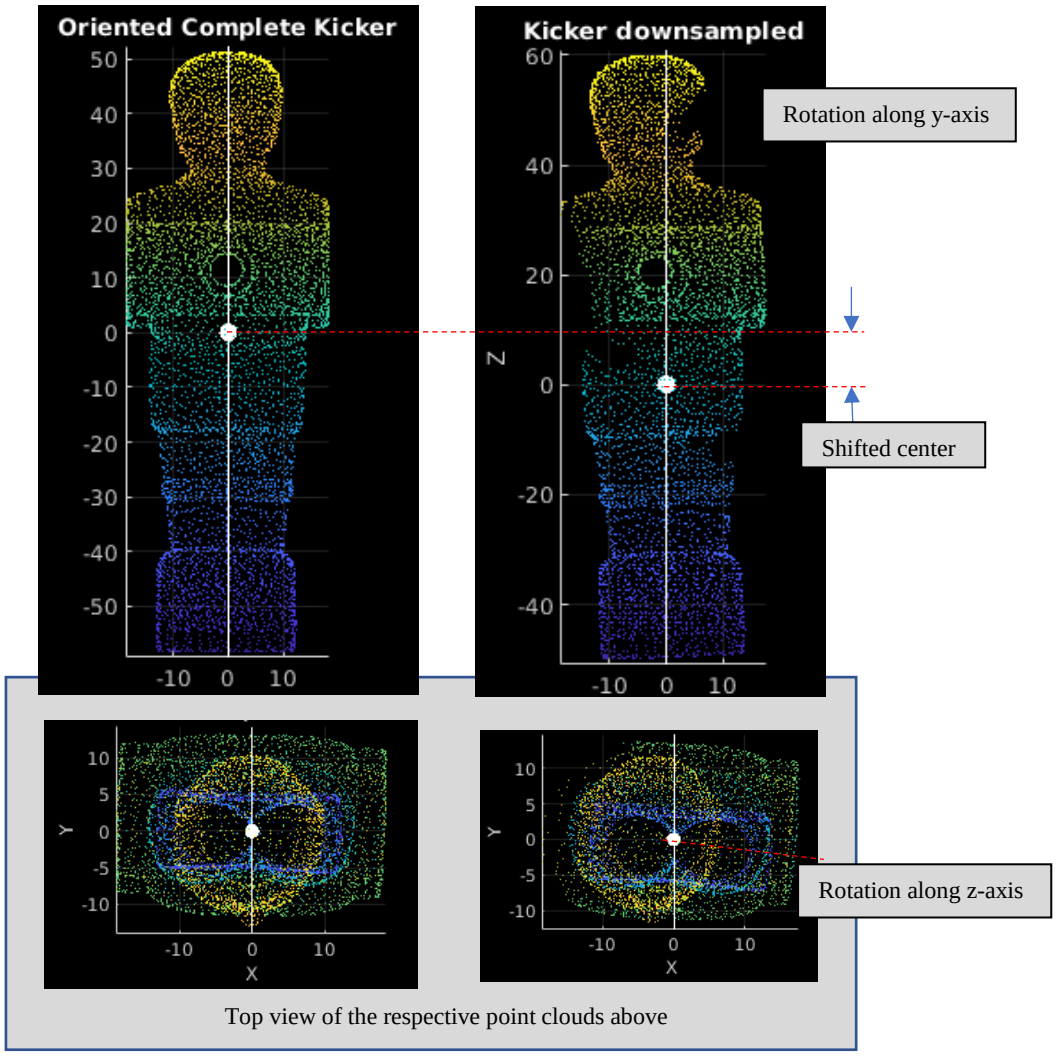



*Figure 25 Comparison of the complete and incomplete point clouds*

- As shown in the figure above, the rotation along y-axis and the shift in the center is clearly visible in the front view, where as the rotation in the z-axis is visible in the top view.

Solution:

The point cloud is translated along z-axis and rotated along y- and z- axes in increments. For each unique combination, the approximately aligned point cloud is oriented to the values in the combination and the root mean square metric is calculated. The minimum of this metric indicates minimum spread of the point cloud.

1. Nested Loops: The pseudocode below describes the methodology used to store all the possible ranges of orientation and their rms metric.

- z-axis translation(z): -5.4 to 9.2 in increments of 0.2 units (20 steps)

  For each z-axis translation:
  - Rotation along y axis (β): 0.25° to 5° in steps of 0.25° (20 steps)

    For each y-axis rotation (γ):
    - Get the Euclidean distance and store with the corresponding z, β and γ values
    - Rotation along z-axis: -0.9° to 1° in steps of 0.1° (20 steps)

      For each y-axis rotation (γ):
      - Get the Euclidean distance and store with the corresponding z, β and γ values

*Figure 26 Pseudocode for logging all possible orientations and rmsmetric values*

- The rms metric is Euclidean distance of the x, y and z value of the points from the approximate center.

$$rmsmetric = \sum_{i=1}^{no.of\ rows} \sqrt[2]{30 * {p_{x,i}}^2 + {p_{y,i}}^2 + {p_{z,i}}^2}$$

$$Where\ p_i : the\ ith\ point\ of\ the\ point\ cloud$$

- For the 8400 unique combinations of offset values, the minimum of the rms metric is found. The corresponding index of the offset list gives the best orientation of the point cloud.

Index of minimum deviation
```
[~, idx] = min(result);
```
Distance to be moved in z-axis to obtain minimum deviation
```
dist = offsets(idx, 1)
```
```
dist = 8.8000
```
Angle β for minimum deviation
```
oy = offsets(idx,2)
```
```
oy = 3
```
Angle γ for minimum deviation
```
oz = offsets(idx,3)
```
```
oz = 1
```

*Figure 27 Orientation values for achieving minimum deviation*

2. Orienting the point cloud: The point cloud is oriented with the minimized values obtained above. The point cloud is translated by 8.8 units in z-axis, rotated in the y-axis by 3° and in z-axis by 1°.

```
[num_rows, ~] = size(points_downsampled);
points_downsampled(:,3) = points_downsampled(:,3) + ones(num_rows,1) * dist;
points_rotated_min = transformPC(points_downsampled , "y", oy);
points_rotated_min = transformPC(points_rotated_min , "z", oz);
```

*Figure 28 Orienting the point cloud with the minimized offset values*

3. Interpolation: The oriented point cloud is interpolated by using the same logic used in section 4.1.4.2 to obtain a point cloud with minimum deviation. The interpolated point cloud is downsampled to remove the extra points.

4. Visualization and inferences: The final interpolated point cloud is visualized and inferences are drawn. The area of incorrect interpolation is highlighted.

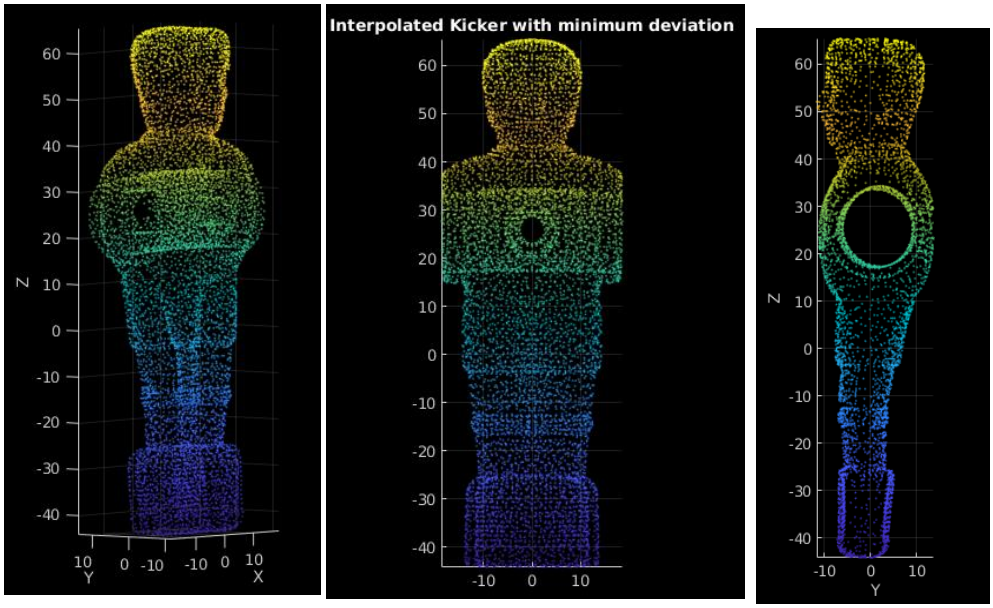



*Figure 29 The final point cloud inerpolated without any registeration functions*

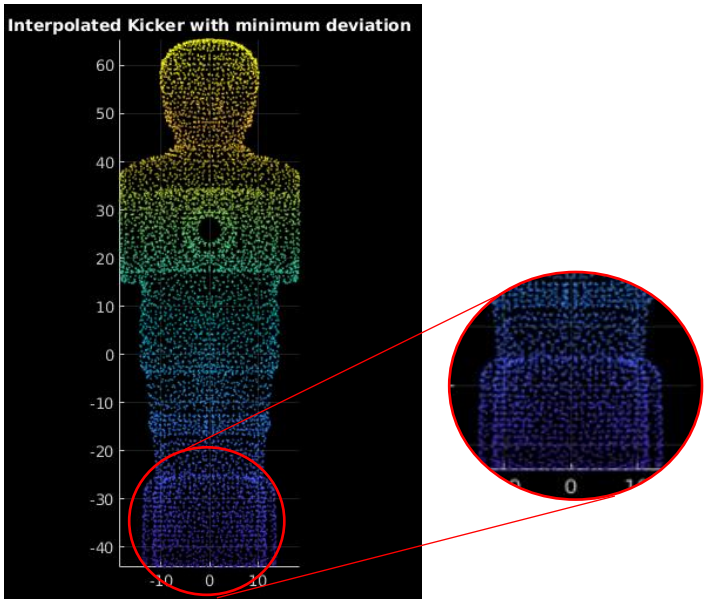



*Figure 30 Incorrect interpolation of an area of the point cloud*

# 5 Discussions

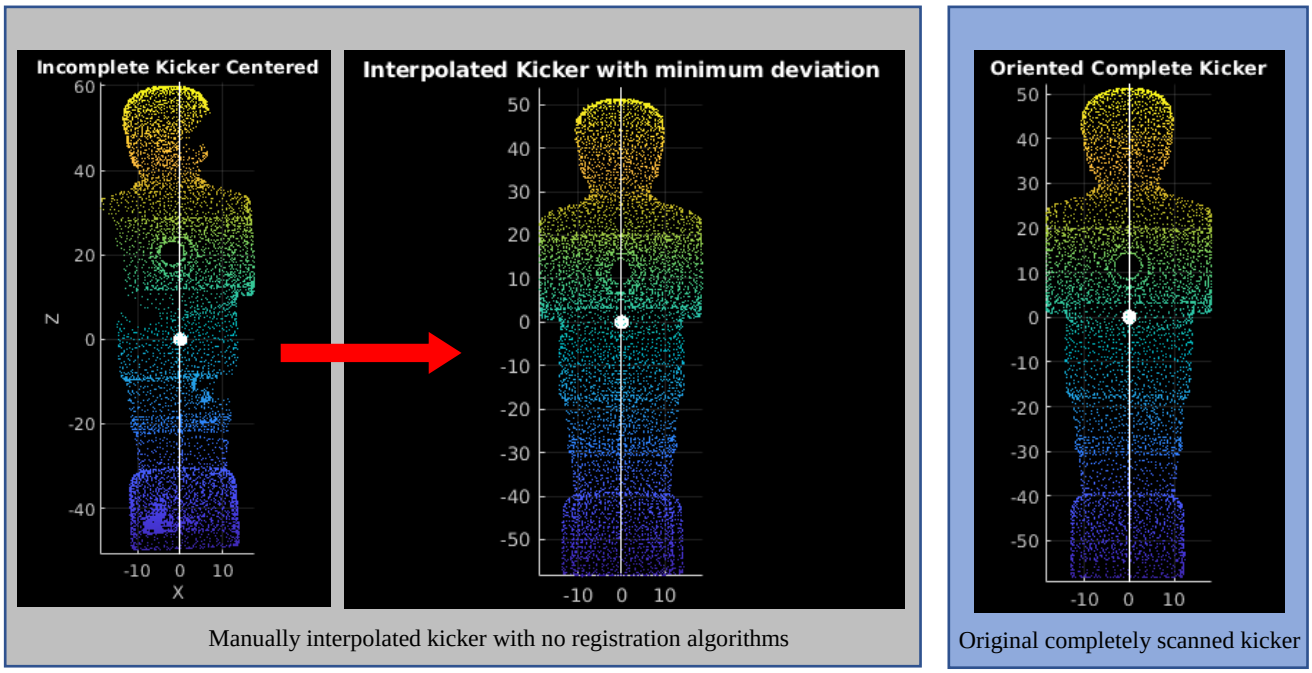



*Figure 31 Side-by-side view of manual interpolation (left) and fully scanned point cloud (right)*

- With the above visualization, it can be concluded that the manual procedure to interpolate the missing points of an incomplete point cloud produces satisfactory results. The overall profiles and features are still preserved.
- A small widened region could be seen in the bottom part of the interpolated kicker. This error can be associated with a coarse step size in the loop. Even though the profiles and details are preserved, there is still some smudging effect visible. The *'pcregistericp'* algorithm does a much better job in preserving the details.
- Even though the inbuilt *'pcregistericp'* algorithm does a much better job in registering the point cloud and producing a near to perfect interpolation, it requires a reference point cloud to produce these results.

- The manual interpolation can be applied with no help of a reference point cloud only when approximate bilateral symmetry plane does not deviate too much from the actual plane.
- When the geometry is too complex or a large number of points are to be interpolated, the manual interpolation might not yield results which are as good as the achieved results.
- The time complexity increases significantly with refined search for the best orientation. In the scope of this project, achieving ~8000 combinations of orientations took around 600 seconds. By refining the step size, the complexity increases linearly. But when one more loop is added, for example the deviation in x-axis (α), the time complexity increases by the power of 2.

# 6 Future Scope

- Even though the algorithm was able to yield satisfactory results, there is still room for improvement.

- A much more efficient grid search algorithm can be implemented to reduce the time complexity of the orientation calculation.

- Yet there are sophisticated inbuilt algorithms in MATLAB to register point clouds, the manual interpolation can come in handy where there are no reference point clouds to register to. When the geometry or point distribution of the point clouds are relatively simple, this project illustrates that with manual interpolations a near to true interpolation is possible.

# 7 Core user defined functions

1. transformPC: Used to rotate a point cloud automatically in a loop.

transformPC.m*

```
function pc = transformPC(ptcl, axis, angle )

%INPUT: A 3d point array/ Pointcloud object;
%       Desired axis of rotation
%       Angle of rotation

%OUTPUT: Rotated point cloud


%Check if the input is a point cloud
t = isa(ptcl, 'pointCloud');
if ~t
    %If not, convert the array to point cloud object to rotate the point
    %cloud using pctransform function
    ptcl = pointCloud(ptcl);
end
%Convert the degrees into radians
angle = -angle*pi/180;


if axis == "x"
    alpha = angle;
    %Rotation Matrix for alpha (rotate along x-axis)
    Rx = [     1            0           0         0
               0        cos(alpha)  -sin(alpha)    0
               0        sin(alpha)   cos(alpha)    0
               0            0           0         1];
    %Build a rigid transformation matrix
    tform = affine3d(Rx);
    %Rotate the point cloud
    pc = pctransform(ptcl, tform);

elseif axis == "y"
    beta = angle;
    %Rotation Matrix for beta (rotate along y-axis)
    Ry = [ cos(beta)   0      sin(beta)    0
               0       1         0         0
          -sin(beta)   0     cos(beta)     0
               0       0         0         1];
    %Build a rigid transformation matrix
    tform = affine3d(Ry);
    %Rotate the point cloud
    pc = pctransform(ptcl, tform);

elseif axis == "z"
    gamma = angle;
    %Rotation Matrix for gamma (rotate along z-axis)
    Rz = [ cos(gamma)  -sin(gamma)    0      0
           sin(gamma)  cos(gamma)     0      0
               0           0          1      0
               0           0          0      1];
    %Build a rigid transformation matrix
    tform = affine3d(Rz);
    %Rotate the point cloud
    pc = pctransform(ptcl, tform);

end
```

2. interpolatePoints: Used to interpolate points of the rotated point cloud and calculate the spread of the points.

interpolatePoints.m

```
function [pc, vals] = interpolatePoints(points)

%INPUT: 3D array of points
%OUTPUT: interpolated point cloud; Euclidean distance of all the points of
%the point cloud to the center of the point cloud. Biased in X-axis


[num_rows,~] = size(points);

    vals = 0;

    %Interpolate points
    for k = 1:num_rows
        point = points(k,:);
        point(1) = -1 * point(1);
        points = [points
                  point];

    end

    %Convert to point cloud object and downsample
    pc = pointCloud(points);
    pc = pcdownsample(pc, "gridAverage", 1);
    pc = pc.Location;
    [num_rows, ~] = size(pc);

    %Calculate the sum of Euclidean distance of all points biased in x-axis
    for k = 1:num_rows

        vals = vals + sqrt( 30* abs(pc(k,1))^2 + ...
                        abs(pc(k,2))^2 + abs(pc(k,3)^2));
    end

    pc = pointCloud(pc);

end
```

3. rotatePointCloudAlongZ: This code was adapted from MATLAB File Sharing web page. The link is in the references section. [8]

4. The for loop used in the project is mentioned below.

```
%Translate the point cloud in z-axis (20 steps)
for z = 5.4:0.2:9.2
    %For each new translation, consider a fresh version of the point cloud
    points_rotated = points_downsampled;
    %Translate along z-direction
    points_rotated(:,3) = points_rotated(:,3) + z*ones(num_rows, 1);

    %Rotate the point cloud in y-axis (20 steps)
    for beta = 0.25:0.25:5
        points_rotated_beta = transformPC(points_rotated, "y", beta);
        points_rotated_beta = points_rotated_beta.Location;
        %Get the Euclidean distance(rms) of all points to the center
        [~, vals] = interpolatePoints(points_rotated_beta);
        %Store the translation, rotY, rotZ values in an array, 'offsets'
        %'offsets' and corresponding Euclidean distance, 'vals'
        offsets = [offsets;[z beta gamma]];
        %Append the rms cvalue into an array
        result = [result ; vals];
        %Rotate the point cloud in z-axis (20 steps)
        for gamma = -0.9:0.1:1
            points_rotated_gamma = transformPC(points_rotated, "z", gamma);
            points_rotated_gamma = transformPC(points_rotated_gamma, "y", beta);
            points_rotated_gamma = points_rotated_gamma.Location;
            %%Get the Euclidean distance(rms) of all points to the center
            [~, vals] = interpolatePoints(points_rotated_gamma);
            %Store the translation, rotY, rotZ values in an array,
            %'offsets' and corresponding Euclidean distance, 'vals'
            offsets = [offsets;[z beta gamma]];
            %Append the rms cvalue into an array
            result = [result; vals];
        end

    end
end
```

# 8. References


[1] MATLAB, "MathWorks," n.d. [Online]. Available: https://de.mathworks.com/help/vision/point-cloud-processing.html.

[2] Wikipedia, "Photogrammetry - Wikipedia," n.d. [Online]. Available: https://en.wikipedia.org/wiki/Photogrammetry.

[3] D. R. R. K. M. B. Alexander Bronstein, "Full and Partial Symmetries of Non-Rigid Shapes," *Full and Partial Symmetries of Non-Rigid Shapes*, p. 23, 2010.

[4] Wikipedia, "Translational symmetry," n.d. [Online]. Available: https://en.wikipedia.org/wiki/Translational_symmetry.

[5] Wikipedia, "Reflection symmetry," n.d. [Online]. Available: https://en.wikipedia.org/wiki/Reflection_symmetry.

[6] MATLAB, "MathWorks," n.d. [Online]. Available: https://de.mathworks.com/help/vision/ref/pointcloud.html.

[7] MATLAB, "Create table from file- MATLAB readtable," n.d. [Online]. Available: https://de.mathworks.com/help/matlab/ref/readtable.html.

[8] K. S. Chan, "Align/Rotate Point Cloud Along Z direction based on PCA - File Exchange - MATLAB Central," Matlab, 24 08 2020. [Online]. Available: https://de.mathworks.com/matlabcentral/fileexchange/79472-align-rotate-point-cloud-along-z-direction-based-on-pca. [Accessed 08 11 2021].

[9] MATLAB Documentation, "Downsample a 3-D point cloud - MATLAB pcdownsample," n.d. [Online]. Available: https://de.mathworks.com/help/vision/ref/pcdownsample.html.

[10] MathWorks, "Register two point clouds using ICP algorithm - MATLAB pcregistericp," n.d. [Online]. Available: https://de.mathworks.com/help/vision/ref/pcregistericp.html.

[11] F. C. R. S. François Pomerleau, "A review of point cloud registeration algorithms for mobile robotics," pp. 1, 44, 27 05 2015.

[12] R. Panigrahy, "An Improved Algorithm Finding Nearest," Microsoft Research, Mountain View CA, USA, n.d.

[13] MathWorks, "Merge two 3-D point clouds - MATLAB pcmerge," n.d. [Online]. Available: https://de.mathworks.com/help/vision/ref/pcmerge.html.

[14] K. Latham, "Radial vs Bilateral Symmetry," 13 03 2021. [Online]. Available: https://biologydictionary.net/radial-vs-bilateral-symmetry/.