The University of Melbourne
Department of Mechanical Engineering

MCEN90032 Sensor Systems - Workshop 3.a

LIDAR Processing

Tuan Khoi Nguyen (1025294)

Introduction

This report will attempt to perform different operations on LIDAR data, which includes setting up a sample LIDAR data and storing it in the point cloud data structure, applying a transformation to the data and try to estimate it back, and run a classification model on different extractions of a LIDAR dataset. These processes will have its methodology described, and will have its effectiveness evaluated through procedural results.

Key goals

The report will include the following listed:

- Constructing a simple LIDAR data.
- Storing the LIDAR data to the point cloud data structure.
- Construct a 3D transformation to be applied to the LIDAR data.
- Attempt to estimate back the transformation using an iterative algorithm.
- Overview on a LIDAR dataset.
- Feature extraction on the dataset using different methods and information.
- Perform a stratified Hold-out to split the dataset into train and test data.
- Classification of the test data using different sets of features, and their evaluation.

Stage 1

Point Cloud structure for LIDAR Data

This stage describes the process of forming a sample LIDAR data, and store it in a MATLAB data structure called `pointCloud`.

1.1 Formation

The requirement states the sample data is in the from of the following equation, where x, y, z denotes the Cartesian coordinates of the data:

$$z = \begin{cases} x^2 + y^3 & 0 \leq x \leq 1\ ,\ 0 \leq y \leq 2 \\ 0 & otherwise \end{cases}$$

To ensure all regions are visualised, other than the main boundaries of interest, an extra 0.1 margin on each side is added to show the effects of the outside region as well. This makes the sampling region becoming $-0.1 \leq x \leq 1.1$ and $-0.1 \leq y \leq 2.1$.

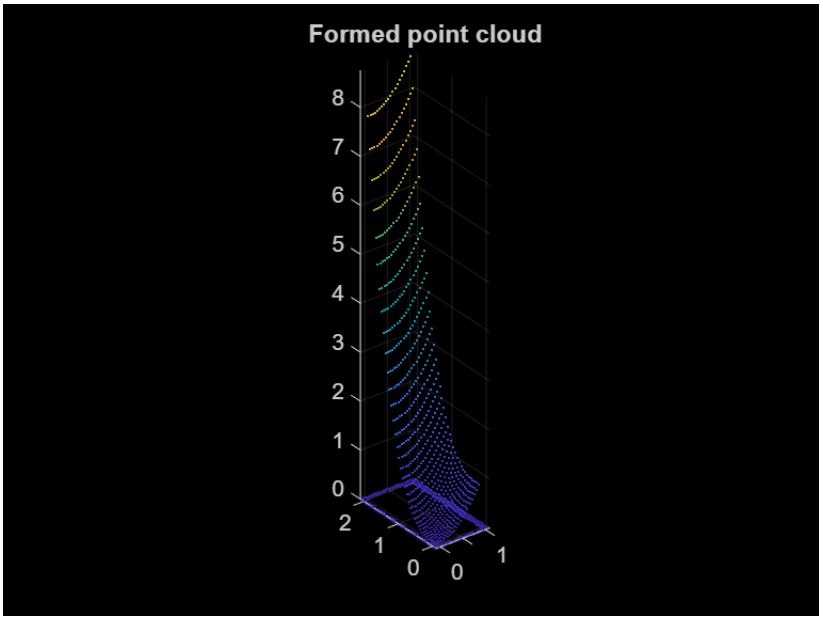
The sample point will be distributed uniformly across the ranges above. To satisfy the requirement of 1000 sample points in the data, the distribution will be set so that there are 25 points in each x bin, and 40 points in each y bin. While this choice does not make perfect uniform unit between 2 axes, the difference would be small, hence making the distribution choice being feasible on uniform data creation.

1.2 Result

```
pc =
  pointCloud with properties:

    Location: [1000×3 double]
    Count: 1000
    XLimits: [-0.1000 1.1000]
    YLimits: [-0.1000 2.1000]
    ZLimits: [0 8.7496]
```

(a) Data Structure description



(b) Visualisation of the point cloud in 3D space

Figure 1.1: Specifications and visualisation of the created point cloud

As shown on Figure 1.1, the data structure satisfied the number of sampling points requirement, with all value regions shown and reflected correctly on the visualisation. The data is now ready for the transformation in the next part.

Stage 2

Transformation and Point Cloud Register

This stage will try to transform the data, by constructing the transformation matrix given the following specifications: rotate 60^o around z-axis, move 2 units in x-axis and 4 units in y-axis. Then, both samples will be used to backtrack the transformation matrix, which will have its efficiency evaluated by comparing with the formed ground truth.

2.1 Transformation Matrix

This section describe the process of forming the transformation matrix to be applied to the point cloud.

Rotation about z-axis

To form the rotation matrix, the following equations are evaluated:

- For a 2D rotation of θ from (x_1, y_1) to (x_2, y_2), using trigonometry will obtain the new elements as follows: $x_2 = x_1 \cos_\theta - y_1 \sin_\theta$ and $y_2 = x_1 \sin_\theta + y_1 \cos_\theta$.
- For rotation around z-axis, the z coordinate will not change: $z_{new} = z_{old}$.

Combining the above together, a system of the following matrix equation is made for θ rotation of a 3D Cartesian coordinate (x, y, z), showing the z-rotation matrix $\mathbf{R_z}$:

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} \cos_\theta & -\sin_\theta & 0 \\ \sin_\theta & \cos_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \longrightarrow \mathbf{R_z} = \begin{bmatrix} \cos_\theta & -\sin_\theta & 0 \\ \sin_\theta & \cos_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation

The specification requires the sample to be moved by 2 units in the x-axis and 4 units in the y-axis. This makes the 3D translation matrix becomes $[2, 4, 0]$.

Combining

With both rotation and translation matrices retrieved, they need to be combined into a transformation matrix to ensure that all steps are applied to the sample in 1 operation. To do so, we can model the state in homogeneous coordinates. The transformation matrix will be 4×4, with the rotation matrix on the top-left, and the translation vector being the fourth row, which will then becomes:

$$\mathbf{T} = \begin{bmatrix} \cos_\theta & -\sin_\theta & 0 & 0 \\ \sin_\theta & \cos_\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 4 & 0 & 1 \end{bmatrix}$$

Substituting $\theta = 60^o$, the matrix now becomes:

$$\mathbf{T} = \begin{bmatrix} 0.5 & -0.866 & 0 & 0 \\ 0.866 & 0.5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 4 & 0 & 1 \end{bmatrix}$$

This matrix will be obtained after putting both the rotation matrix and translation matrix into the MATLAB built-in function `rigid3d()`, which returns a transformation data structure with the same name. The transformed data will then be obtained after applying this data structure to the point cloud, using the built-in function `pctransform()`.

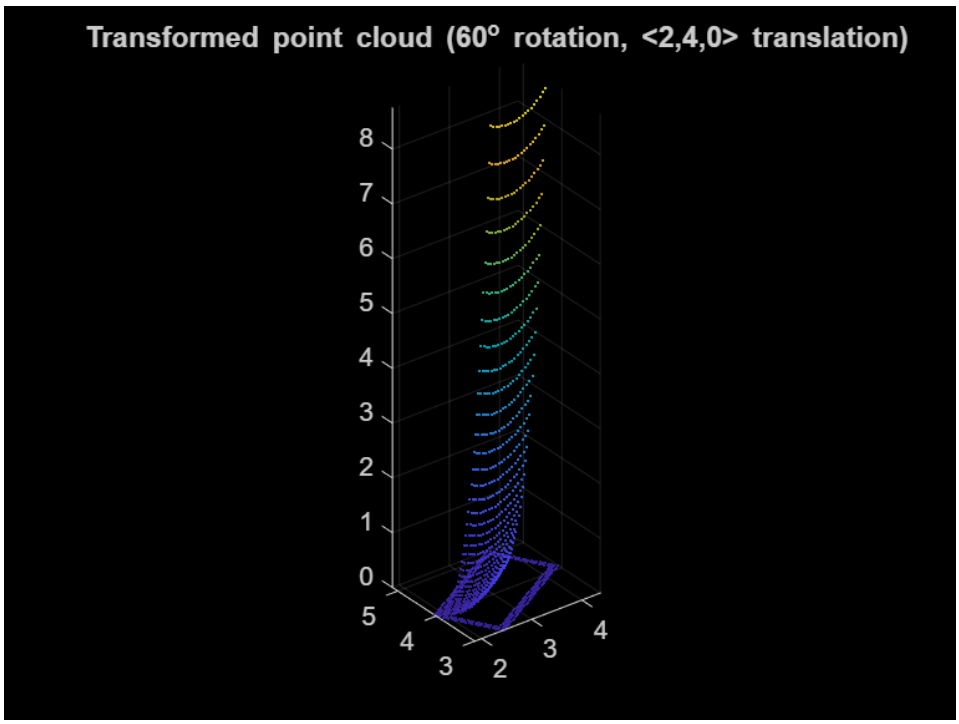


Figure 2.1: Result of applying the transformation to the created point cloud

In the next section, an iterative method will be used to estimate the transformation matrix using only these 2 datasets.

2.2 Estimation of translation

2.2.1 Iterative Closest Point (ICP) Algorithm

ICP can estimate the transformation of the data, by applying its guessing transformation to the original data, compare the result against the transformed data, and update the guess accordingly [1]. The procedure for ICP can be described in details as follows:

- Input initial guess of the transformation $\hat{\mathbf{T}}_\mathbf{0}$, which consists of a rotation and a translation.
- Iteratively do the following at step i until convergence or iteration limit reached:
  - Apply the guessed transformation into the data: $\hat{d}_t = \hat{\mathbf{T}}_\mathbf{i} d_0$
  - Compare the transformed data $\hat{d}_t$ against the true transformed data d_t, by getting the closest point in d_t to each point in $\hat{d}_t$.
  - Using Least Square Approximation, obtain the next guess $\hat{\mathbf{T}}_{\mathbf{i+1}}$.

To perform ICP on the 2 samples, the MATLAB function `pcregistericp()` is used. This function has the following parameters of interest:

- `Tolerance = [0.01,0.01]`: Respectively, the smallest update allowed for translation vector and the rotation matrix, which Euclidean distance being the criteria. If an update step does not move far away from the last iteration, the function will assume that the estimation has converged. It should be small enough to have a precise result, but large enough so that the iteration process will not take too many runs to reach.
- `MaxIterations = 100`: The maximum number of iterations before stopping the algorithm. The number should be set to a large enough value to ensure the error converges, but should be small enough so that the computation process would not take too long if the error diverges.
- `Verbose = True`: For each iteration, prints out the relevant information. In this task, the information of interest will be the number of iterations and the error throughout each iteration.

In this function, the root mean squared error (RMSE) of sample points is the criteria to assess the error of the true sample.

2.2.2 Results & Parameter Effects

```
true_transform = 4×4
    0.5000   -0.8660         0         0
    0.8660    0.5000         0         0
         0         0    1.0000         0
    2.0000    4.0000         0    1.0000

Total number of iterations: 26

predicted_transform = 4×4
    0.5000   -0.8660         0         0
    0.8660    0.5000         0         0
         0         0    1.0000         0
    2.0000    4.0000         0    1.0000
```

Figure 2.2: MATLAB outputs of the transformation matrices: ground truth and predicted using ICP

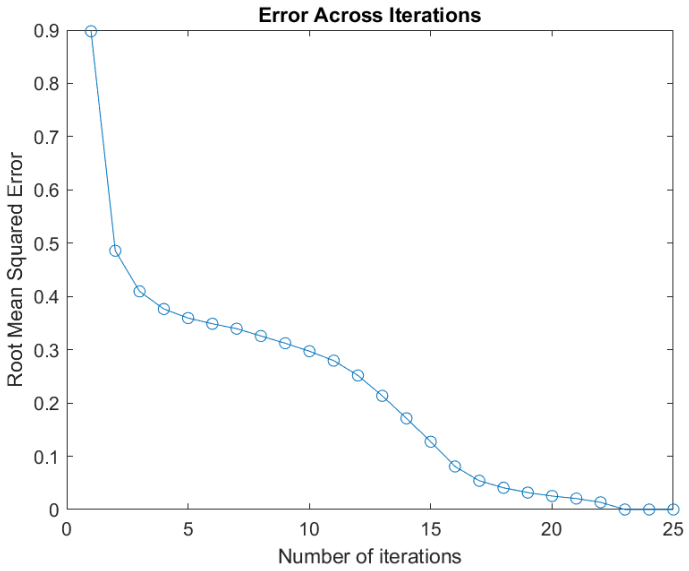


Figure 2.3: The RMSE of the sample points converging through each iteration

The outputs in Figure 2.2 and 2.3 shows that the RMSE converging to 0 in 26 iterations, proving that ICP algorithm with the defined parameters in the previous section is efficient enough to use on the given set of data. With transformations being found, LIDAR samples will be able to be united into specific coordinates and orientation, which is potentially useful for object recognition tasks.

Stage 3

LIDAR Object Recognition

This stage will attempt to perform object recognition using multi-class Support Vector Machines (SVM) and test the model on different set of features extracted on the data. Next, the efficiency of each dataset will be evaluated to see how well can different features help in categorizing objects.

3.1 Data Description & Features

The dataset contains 523 instances of samples, each containing an arbitrary number of sampling points. Each point has 4 attributes: the 3 Cartesian coordinates of the point in 3D space, and a value denoting its intensity. Due to different camera specifications or environmental factors such as lighting or fog, samples of the same object may have varied locations or reflections, resulting in different values of the attributes above, which will be hard to track if sampling condition is not known. Therefore, other features are extracted based on these attributes in order to produce more consistent features across the same category. The following sections will describe the information chosen to be extracted from the dataset.

3.1.1 Intensity-based features

Across the scanning range, the intensity of each sampling point may vary. Therefore, estimating its distribution would give better information to determine what an object is. For this task, the following 3 statistical features will be used to describe the intensity distribution:

- Mean: shows what range of values appears the most.
- Standard deviation: shows the spread of the distribution.
- Median: shows how skewed the distribution is.

3.1.2 Spread-based features

While locations of the object may vary in the sampling area, its shaping should remain the same. To measure this, it is feasible to track its size. Therefore, for each axis, spread - the furthest distance between the 2 data points, will be tracked, and used in combination with the intensity-based features.

3.1.3 PCA-extracted features

Principal Component Analysis (PCA) is a method that tries to extract the most information from the data while keeping the dimension minimal, by extracting the principal components of the data. The steps for PCA are done as follows:

- Project the data and get its covariance matrix.
- Get the principal component that covers the most significant data relationships, which is reflected by the eigenvector corresponding to the largest eigenvalue in the covariance matrix.

To get similar behavior like the original spread features, after retrieving the principal components from PCA, their variances will be taken and set as the PCA-extracted feature. The MATLAB built-in function `pca()` is used for this part.

3.2 Classification Setup

3.2.1 Stratified Hold-out

For this task, the dataset is separated using the 30-70 Hold-out method: 70% of the data will used to train the classification model, which will then be tested on the other 30% of the data. Stratification is used, which will try to ensure that the partitions will have similar label distributions. This is further confirmed by checking if all labels appear in both the training and testing dataset.

For this process, the MATLAB built-in function `cvpartition()` is used, with the following parameters defined:

- `HoldOut = 0.3`: Specify that Hold-out is used, with 0.3 of the data will be used as testing data.
- `Stratify = true`: Do stratification when splitting the dataset.

3.2.2 SVM Parameters

The classification model used in this task will be multi-class SVM with linear kernel, where for any 2 classes, it tries to find the line that best separate them, which will have the largest margin between classes, and set it as the boundary to distinguish them [2]. The multi-class model will use one-versus-one strategy, where it will run multiple binary SVM classifiers across pairs of labels and take the majority among these predictions. To ensure consistency in this progress, the data will also be standardized to have the same mean and standard deviation. For this process, the MATLAB built-in function `fitcecoc(data, Learner = templateSVM(...))` is used, with the following parameters defined in `templateSVM()`:

- `KernelFunction = Linear`: Specify that the kernel function used in SVM will be the Linear Kernel.
- `Standardize = true`: Perform standardization on all datasets before training or prediction.

3.3 Results & Comparison

For comparison between features, accuracy - the proportion of correct predictions, will be used as the assessing criteria of interest. The baseline 0R - proportion of the most common label in the dataset, will be used as the minimum accuracy requirement to see if the model has reasonable performance.

Features	Accuracy
0R Baseline	29.06%
3 intensity features	50.64%
3 intensity features + 3 spread features	80.13%
3 intensity features + 1 PCA extracted feature	69.23%
3 intensity features + 2 PCA extracted features	75.00%
3 intensity features + 3 PCA extracted features	79.49%

Table 3.1: Result of running Linear Kernel SVM on different sets of features

From the result table observed in Table 3.1, the model seems to work best when both intensity and location features are used. When removing the spread features, the accuracy reduced up to around 30%, proving that they provide important information for classification. As the 3 intensity features alone can give about 50% accuracy across 10 different labels and exceeded the minimal baseline, it can also be concluded that they are useful features in this classification process.

For cases where PCA is used to extract the data spread instead of taking the furthest distance between coordinates, they all resulted in at least 20% higher accuracy than using the intensity features alone. The highest among them is when 3 PCA-extracted features are used, which reached 79% accuracy, proving that it can be as useful as the original spread features.

It is also notable that when adding an additional PCA-extracted feature, the more PCA-extracted features existing on the data, the smaller accuracy improvement will be, proving that PCA, as expected, have extracted most of the meaningful information to its top features, making it being optimal on fewer features.

3.4 Conclusion & Potentials

Overall, the multi-class linear SVM performed reasonably well on the given sets of attributes, consisting of intensity distribution features, spread features and PCA extracted features, proving that they are all close to linearly separable, which is meaningful for linear SVM models. PCA-extracted feature have proven to perform as well as original spread features. Each additional PCA feature provides the most information at first, but will have much slower improvement as more of them is added, hence being the most optimal at a finite number of PCA-extracted features. Further ideas that can improve the accuracy further may include using a different SVM kernel function for the data that better matches the data distribution, or use others model that exploits the features differently. These results will be meaningful for expanding into many processes such as object tracking, or intention prediction.

Bibliography

[1] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, September 1987.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

Appendix A

Additional Resources

A.1 Least Square Approximation

Least square method is a method used to estimate a function or group of variables $f(parameters)$, by trying to minimize the errors surrounding it. For this task, for estimated transformed data (ETD) and true transformed data (TTD), the error of interest is the sum of Euclidean distances from one point in ETD to its closest point in TTD:

$$err = \sum_{d \in ETD} ||d - closest(d))||_2^2$$

A common approach is to find the parameters such that the derivative of f is 0.

A.2 Root Mean Squared Error (RMSE)

RMSE is a common method to measure the error of a function. As the name suggests, it is the square root of the mean of squared errors:

$$RMSE = \sqrt{\frac{1}{N} \sum_i |\hat{x}_i - x_i|^2}$$

A.3 One-Versus-One (1v1)

1v1 is a common method to extend a binary classifier to multi-class. It operates by getting all possible unique pairs of the existing labels, and run the binary classifier to try and get the more likely label between the two. For each instance, the majority of these pairwise predictions will be used as the final label.