Northwestern University

ME 469: Learning Comparison

YIKE LING, VISHWAJEET KARMARKAR, & MICHAEL RENCHECK

GMM-GMR Comparison – Yike Ling

Question 1: Learning Aim Description

My partner Michael is using Locally Weighted Linear Regression (LWLR) as a learning algorithm to train his dataset and then make predictions. His learning aim is to predict a change in state using the provided groundtruth data and odometry data, specifically he is looking for position change (dx, dy and dθ).

Question 2: Learning Algorithms Comparison

In LWLR, points are weighted by proximity to the current data point and then regression is computed by those weighted points. Unweighted regression will generate a linear relationship ($y = mx + b$) for all data points but when each point is given different relevance by the formula:

$$wX\beta = wy$$

$where\ \beta\ is\ (n+1)\ \times 1\ size\ martrix\ and\ n\ is\ the\ dimentionality\ of\ inputs,$

$w\ is\ weights\ which\ is\ N \times N\ martirx\ that\ N\ is\ number\ of\ data\ points.$

$X\ is\ a\ N \times (n+1)\ matrix\ and\ y\ is\ a\ (N \times 1)\ matrix.$

By adding weights, LWLR then can be a good estimator for a non-linear function. It will attempt to fit the locally linear model to training data set in in order to have decent estimations for outputs. However, when searching for locally linear model, results can be shifted by outliers and yield unwanted predictions. This can be more obviously to be observed when the number of data points is small if a valid outlier exists. In addition, LWLR can be considered as a lazy learning because there is no training involved before inputs and it is a supervised learning method because it uses training data with known outputs.

GMM-GMR (Gaussian Mixture Models & Gaussian Mixture Regression) implementation has two parts. GMM is a clustering algorithm which uses Gaussian distributions. Commonly the approach is using Expectation & Maximization (EM) to find Gaussian states parameters. EM is an iterative algorithm that converges and stops by log likelihood threshold which the equation is listed below:

$$logL(\theta) = \sum_{j=1}^{N} \log\left(\sum_{k=1}^{N} \alpha_k \phi(x|\theta_k)\right)$$

$where \sum_{k=1}^{N} \alpha_k \phi(x|\theta_k))\ is Gaussian distribution function,$

$\alpha_k is the probability that recording data belong k_{th} Gaussian model,$

$\phi(x|\theta_k) is the Gaussian distribution function of k_{th} Gaussian Model$

Note that in the Gaussian distribution function,$\theta_k = (\mu_k, \sigma^2{}_k)$, which represent mean and covariance of the kth Gaussian model. Each iteration of EM will have a new set of $(\mu_k, \alpha_k, \sigma_k)$ until EM converges and stops. GMM is a training and after GMM is finished, GMR, as a prediction phase will retrieve output features by time input. GMM-GMR is an unsupervised learning since it does not have pre-existing outputs and it only cares about data patterns after clustering and some dissimilarities among data points.

Question 3:Learning Algorithms Applied to the Aims

I implemented GMM-GMR on my partner's dataset. He is using 1000 data points for testing and the rest of his dataset for training. However, because of the problem of my laptop's system, I used part of his training dataset in training phase.

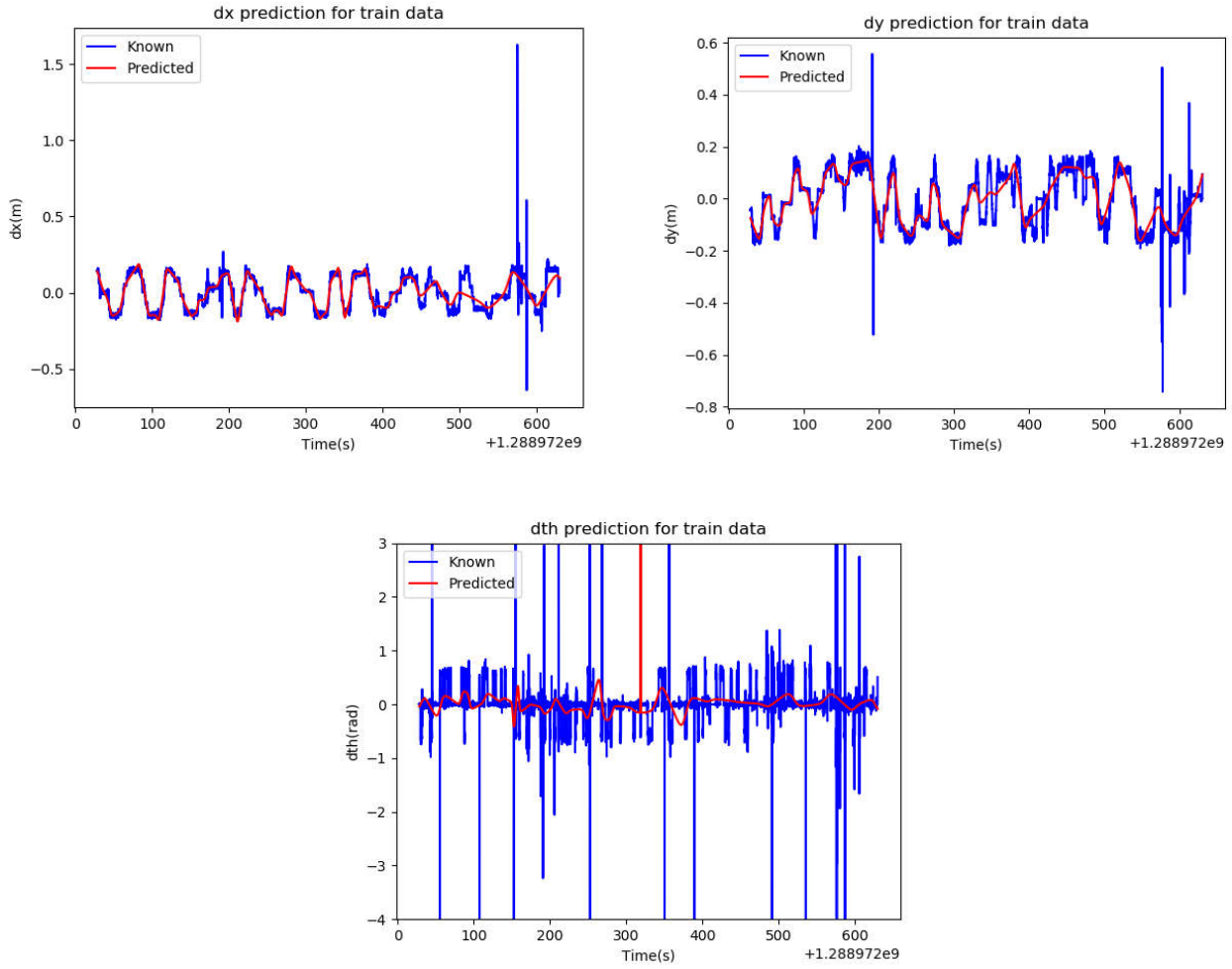


Figure 1. Training setresults forstate changes

GMM can be viewed as a model consisting of multiple Gaussian models and in my algorithm implementation, number of clusters, k needs to be defined. However, the larger k is, the longer training time will be. Thus, it is necessary to make choice of k after observing the un-clustered data first.

As explained previously, single Gaussian model is essential for GMM and probability density can be computed by the formula,

$$P(x|\theta) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{(x-\mu)^T\Sigma^{-1}(x-\mu)}{2}\right.$$

where D *is the dimensionality of dataset,* Σ *is covariance and* μ *is mean*

Probability needs to be computed for every Gaussian model and EM is a popular method to compute (α, Σ, μ).

GMR then will act as a generalization process which computes conditional probability p(x | t) where x is data we intend to train and t is the temporal time value.

5000 data points were selected as training data and red line represents predictions and blue line shows the state change x calculated from groundtruth dataset. By observing resulting plots, we can see that training is good enough for dx and dy but the performance is bad on dθ. k = 55 was chosen to be the number of clusters for Gaussian models. Number of clusters can be increased but is restricted by the system of the laptop. Number of clusters is deciding how data are clustered. When this number is larger, GMM will compute more clusters. If large enough number of clusters is chosen, red line will fit calculated known data points perfectly. The average training time was 1 min. By comparing three plots in Figure 1we find that for dx and dy, the red line fits quite well when the slopes of trajectory of blue are relatively flat.

Next testing data are chosen from Michael's dataset. Index 549 to index 1549 make the testing dataset. 55 was again chosen to be the number of clusters. Results are satisfied by observing that predictions are well following the trend of known data points. The resulting plots are displayed below:

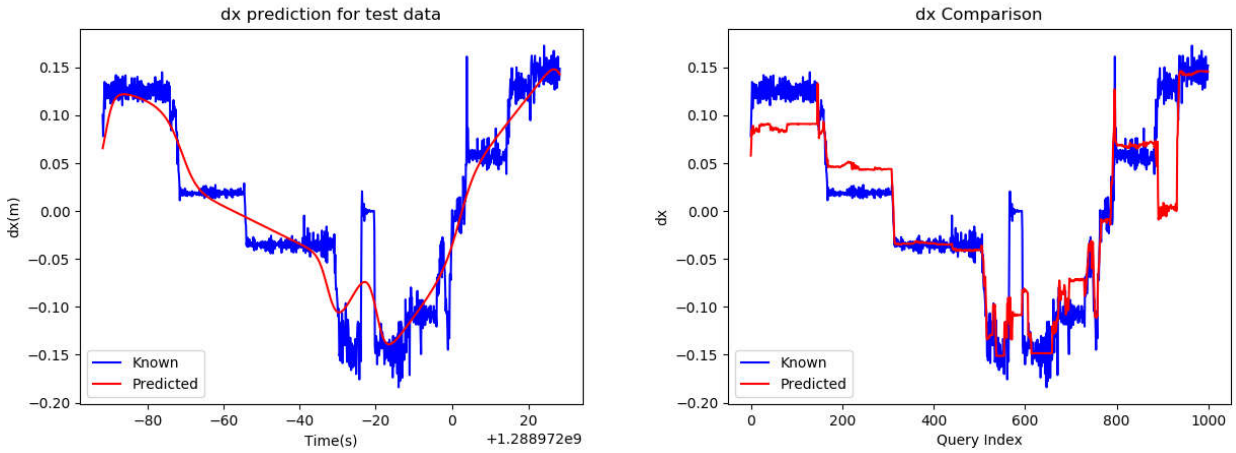


Figure 2. Testing Set of State Change x Comparison

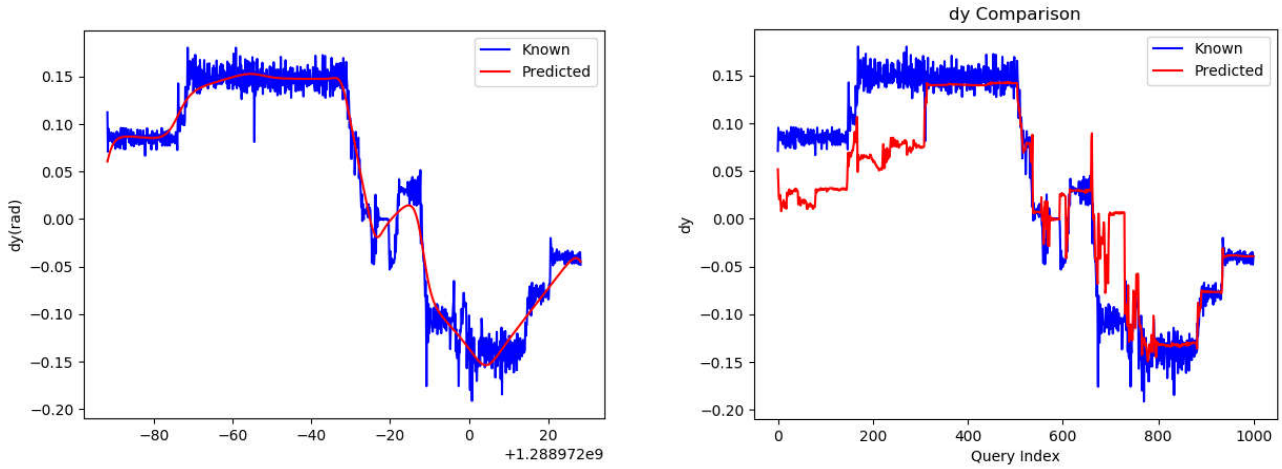


Figure 3. Testing Set of State Change y Comparison

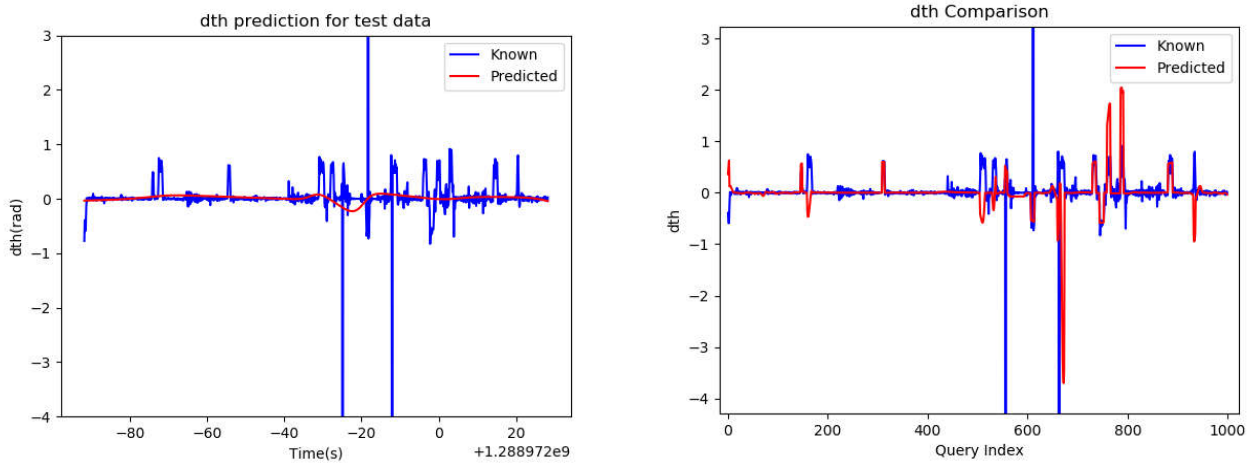


Figure 4. Testing Set of State Change θ Comparison

Question 4: Assessment and Discussion

Figure 2, Figure 3, and Figure 4show comparison between two algorithms after testing chosen 1000 dataset. Both algorithms are performing well in predicting. Predictions are following closely along the known data points. However, for GMM-GMR, the performance is not detailed enough by comparing plots by LWLR and the reasonable explanation is that LWLR is computing the locally linear relationships between all neighbors while for GMM-GMR, since 55 is the number of clusters and that does not yield enough Gaussian models and thus predictions cannot be perfectly made although the overall trend is great. By computing errors, as displayed inTable 1, the performance of two algorithms are close by the average error analysis. R^2 error analysis describes the variability of data point to mean and if the value is 1 or close to 1, the predicted data are fitting well. R^2 results proved that predictions are much better for dx and dy tests for GMM-GMR than Locally Weighted Regression. Although R^2 is -0.00019 and –0.06808 for dθ, the possible reason is that the Gaussian models are not successfully computed for those calculated dθ which are too large comparing to other points and valid outliers are affecting weighted locally linear models.
In the table, error analysis is produced by testing different number of clusters. 55 and 30 result different results. By observing them we can conclude that with higher number of clusters, predictions are better.

Errors have been computed in two ways, R^2 error analysis and average error analysis. Formulas for these two error analysis methods are listed below:

$$\text{Average error analysis: } e = \frac{\sum_i^N |\hat{y}_i - y_i|}{N}$$

where $\hat{y}_i$ *is predicted data point*
y_i *is actual data point*
N *is the totall number of dataset*

$$R^2 \text{ error analysis: } R^2 = 1 - \frac{\sum_i^N (y_i - \hat{y}_i)^2}{\sum_i^N (y_i - \bar{y})^2}$$

where $\hat{y}_i$ *is predicted data point*
y_i *is actual data point*
$\bar{y}$ is the mean of actual dataset
N *is the totall number of dataset*

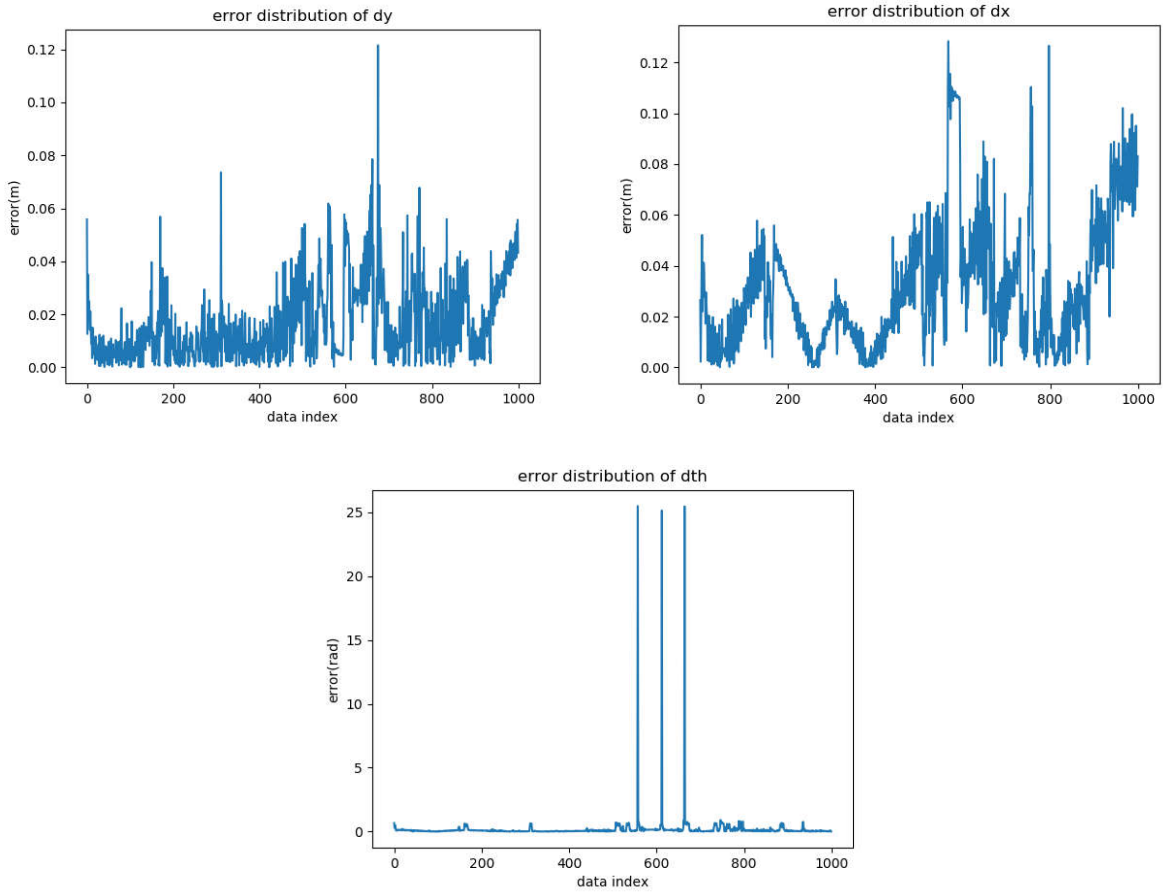


Figure 5. Error Plots for the predicted state change

Table 1. Error Calculation comparison

	Average Error Analysis		
	Error dx	Error dy	Error dθ
GMM-GMR (55)	0.03135 m	0.01647 m	0.1951 m
GMM-GMR (30)	0.04021 m	0.02474 m	0.1897 m
LWLR	0.02625 m	0.03379 m	0.1950 m
	R^2		
	R^2 dx	R^2 dy	R^2 dθ
GMM-GMR (55)	0.8775	0.9585	-0.00019
GMM-GMR (30)	-1.2167	0.9117	-0.00014
LWLR	0.8305	0.8216	-0.06808

Question 5: Conclusion

GMM-GMR and Locally Weighted Regression both work well on predicting the aim but GMM-GMR is slightly better. However, performance can be approved much better for GMM-GMR if a suitable number of clusters is found but generally GMM-GMR cannot predict well if data vary a lot. GMM is trying to cluster and generalize data, thus any small and detailed clustering will require more Gaussian models. On the other hand, LWLR is attempting to compute weighed locally linear relationships so it can analyze detailed relationship between neighbors, but valid outlying points can impact its judgments.

Bibliography

Sylvain Calinon, "On Learning, Representing and Generalizing a Task in a Humanoid Robot", LASA Laboratory - Ecole Polytechnique Federalee de Lausanne (EPFL), CH-1015 Lausanne, Switzerland

Neural Network Comparison – Vishwajeet Karmarkar

Question 1: The New Learning Aim

The old learning aim was to see whether past data from the robot's motion can be used with a neural network to learn the robot's state transition, without us explicitly modelling the physics of the agent and its environment.

The new aim is to improve the Measurement Model by learning from the measurements taken by the robot's sensors. The measurement model is as follows.

$$Range = \sqrt{(x - x_{landmark})^2 + (y - y_{landmark})^2} + \epsilon_{noise}$$

$$Bearing = atan2\left(\frac{y_{landmark} - y}{x_{landmark} - x}\right) + \epsilon_{noise}$$

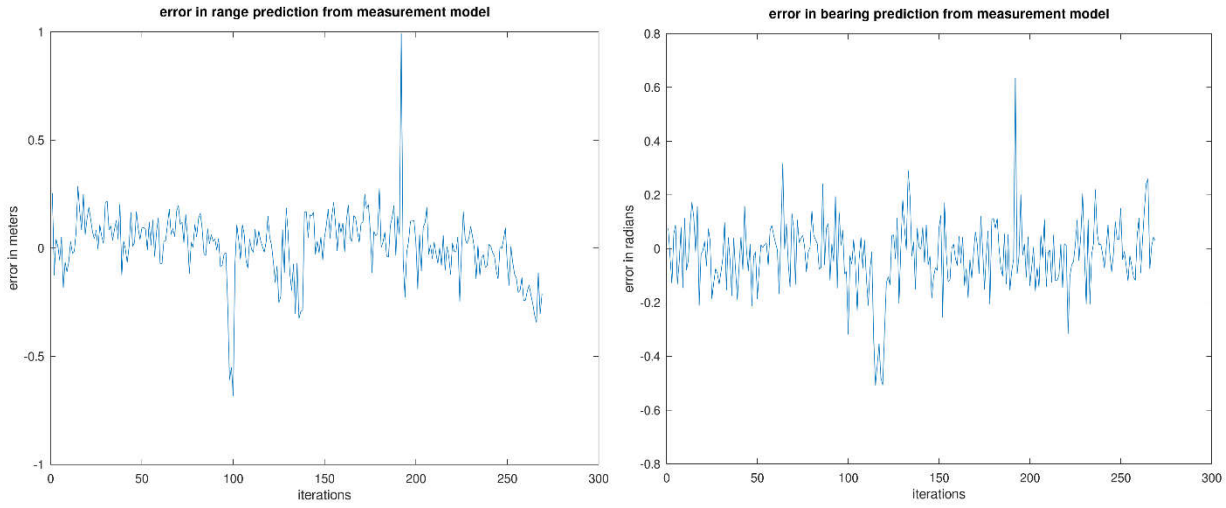


Figure 6. Measurement model error

The two images above show the measurement model error. The values are calculated from the equations shown above, with Gaussian noise having 0 mean and variance values of 0.006 and 0.009 for range and bearing respectively. Assuming that the sensor data and ground-truth locations are correct, the algorithm should learn to closely match the sensor readings, and thus attempt to minimize the error seen above. This helps the robot better estimate its state, thus justifying the exploration of machine learning techniques towards this pursuit.

Question 2: Learning Algorithm Comparison

Neural Networks:

1) NN's fall under the supervised learning category. Thus they need an input vector of training examples and corresponding output labels : [X, Y].
2) Another aspect to NN's is that post training it is not possible to deduce any meaning from the weights and the node activations that are found post training in the various hidden layers. Thus, Neural networks can be looked at as a 'black box', from a human understanding perspective.
3) Neural Networks are a parametric learning model. For parametric models, with new data and thus new relations, parameters may have to be re-tuned (for eg number of layers, nodes, activation function etc).
4) Neural networks scale very well. Deep Networks are being widely used for large datasets. They don't perform that well on smaller data-sets.

GMM-GMR:

1) GMM-GMR's are a form of unsupervised learning. The dataset contains data in the form [t,x] where t is temporal value and x is the feature vector. There are no output labels. Thus, various Gaussian models with a 'mixture component' are fit to the data. GMR helps us predict p(x|t), which is expected x at time t.
2) GMM-GMR outputs maximized values of mean and co-variance. Meaning can be deduced from these Gaussians as they are easier to understand.
3) GMM-GMR is a non-parametric model and thus is flexible enough to accommodate new data (in the same feature space) without needing modifications.
4) These algorithms don't scale very well. They are much more effective with low training examples

Question 3: Learning Algorithms Applied to the Aims

The dataset provided by my partner (GMM-GMR) included [Time, Ground X, Y, Theta, Range, Bearing, Landmark X, Y] in its input space. The range and bearing values in the data are with reference to landmark 7. Since it is an unsupervised learning, there are no output labels. This leads to the first modification of the data for neural networks, with the Range and Bearing as the output labels, while remaining features are kept as input labels. Time-stamps in the input data are of a much greater magnitude compared to other parameters. Thus, the input space was normalized to prevent false importance to the time-stamps. The equation shown below is used to normalize the inputs.

$$X = \frac{X - mn(X)}{SD(X)}$$

$mn = average(X)$
$SD = Standard\ Deviation\ of\ X$

The neural network was designed as a shallow network with 30 hidden units. Batch Gradient descent (with momentum update) was used, with the entire training set as the batch. The number of iterations of weight update, and learning rate were decided by trials. 10,000 iterations were performed for the final program.

Table 2. Example results for NN training

Iterations	R2 value for bearing on testing set
5000	0.64802
10000	0.76932

The learning rate and momentum parameters were tuned by observing the progress of the 'cost' with iterations. Too high learning rates led to a premature halt of the cost reduction, or even reversed the process. Too low learning rates made the process very slow. Examples are shown below.

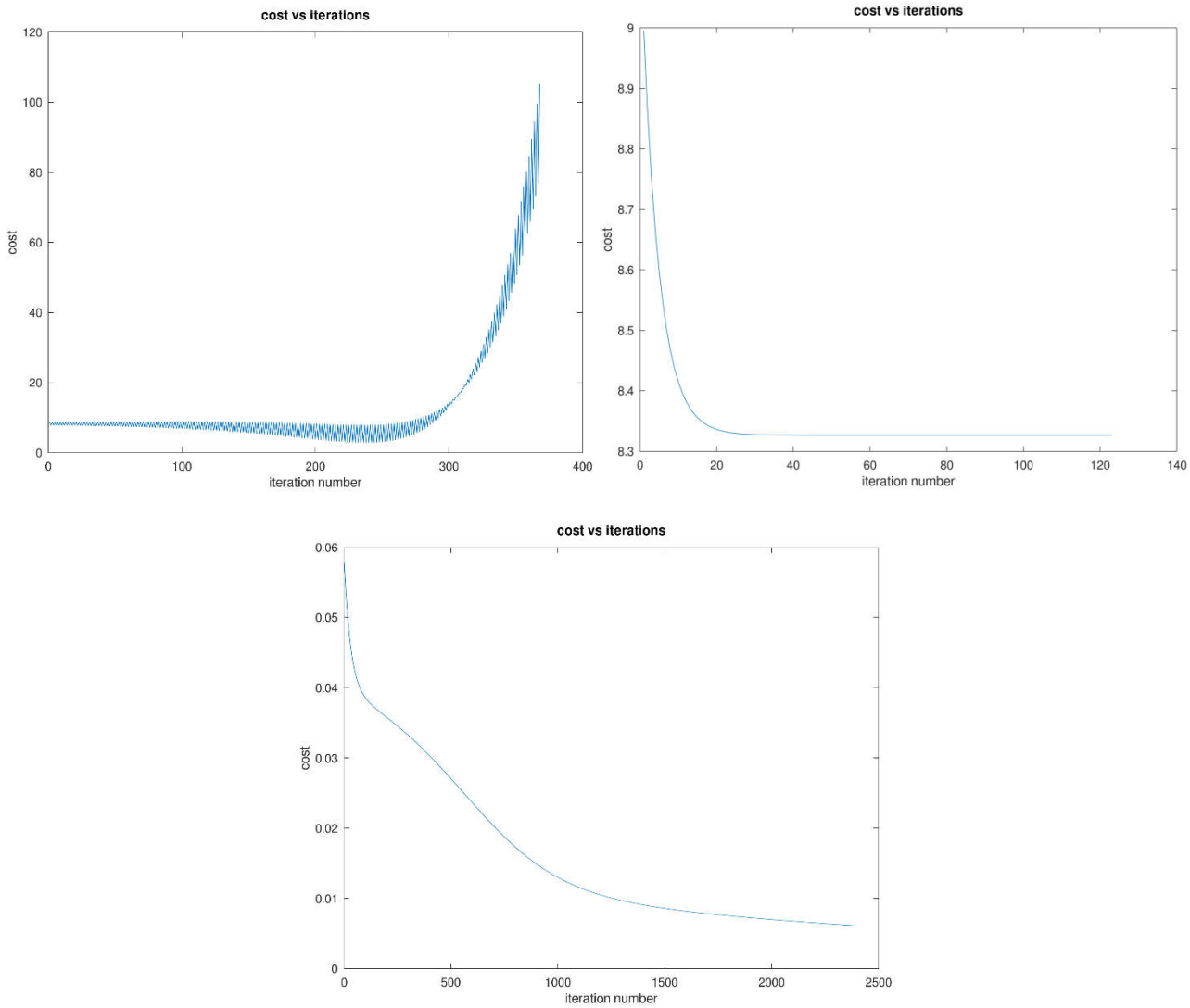


Figure 7. NN Learning rate tuning examples, very-high learning rate (top-left), high-learning rate (top-right), good learning rate (bottom)

As seen in the first example, choosing a very high learning rate leads to oscillations in the cost, and at a certain point it completely reverses, moving away from convergence. For a lower (but still high) learning rate, the system appears to converge, but at a very high cost value, and that too very fast. Choosing a good learning rate leads to continuous cost reduction with a 'spread-out' pattern across iterations. This can be seen in the third image with costs reducing at a good pace up-to 2500 iterations. The pattern continues and is cut-off at 10,000.

Question 4: Assessment Comparison

To compare the performance of the algorithms, range and bearing predictions are referred below.The data-set was split into training and testing (70% - 30%). Post training, the test set is used to evaluate how well the data generalizes to new values, and ensures that the model isn't just 'remembering' the data to output values. The series of images above show the comparison of the performance of GMM-GMR (on the left side) and neural network on the right, for both range and bearing predictions.

It can be observed that in evaluating the prediction for the training data, both GMM-GMR and neural networks are able to perform quite well. There appears no particular advantage in using one of these two algorithms. However the testing data plots lead to a different evaluation. Zoomed in views of a few contrasting predictions are shown below in Figure 10.

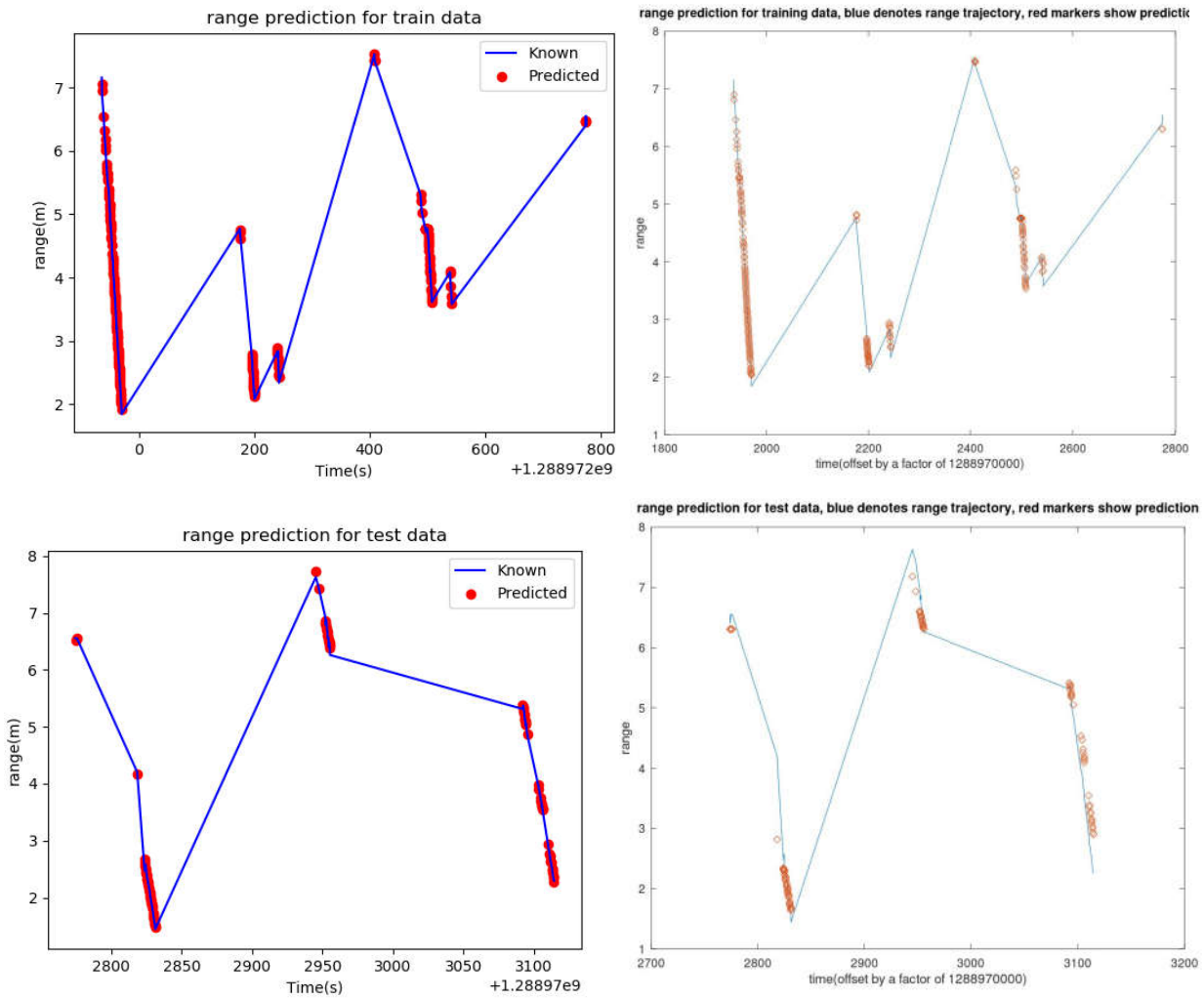


Figure 8. Range prediction comparison for GMM-GMR (left) and NN (right)

The neural network clearly isn't generalizing as well as the GMM-GMR. Another metric to verify this is by looking at the Mean Squared Error and the Co-efficient of determination values. Lower the mean-squared error, better the estimate of the model. R^2 tries to see how well the fit describes the variability of the data about its mean; If R^2 is 100% then that implies that the fit describes the variability perfectly.

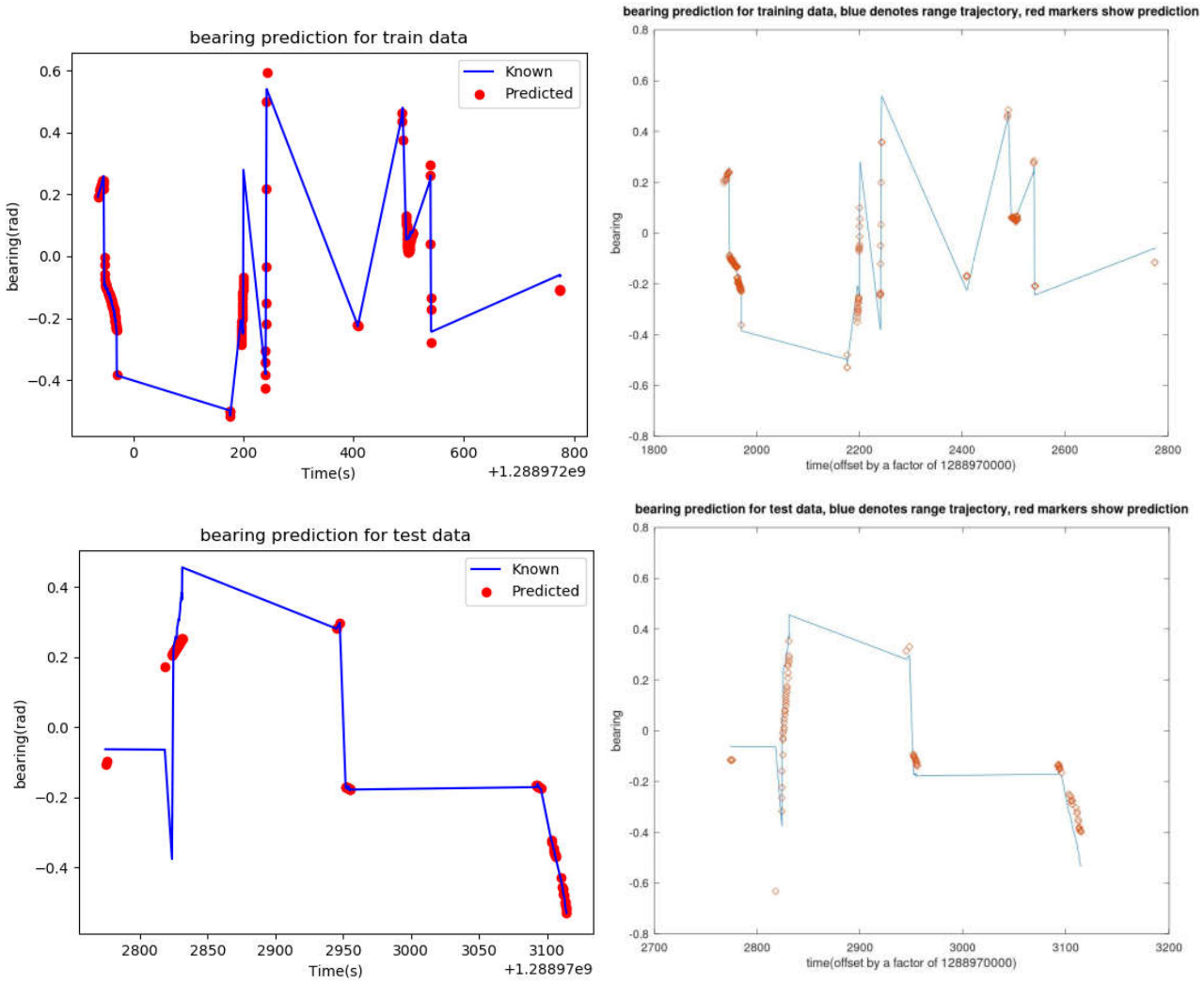


Figure 9. Bearing prediction comparison for GMM-GMR (left) and NN (right)

Table 3. MSE and R^2 comparison for NN and GMM-GMR results

	MSE Training	MSE Testing	R^2 Training	R^2 Testing
Range	0.0061136	0.136504	0.99613	0.96341
Bearing	0.0030871	0.018409	0.91040	0.78819

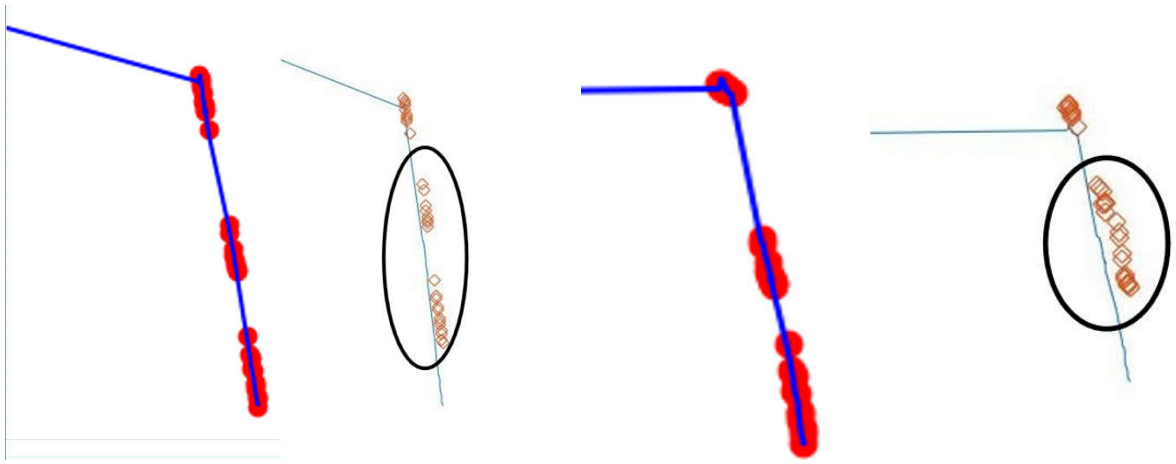


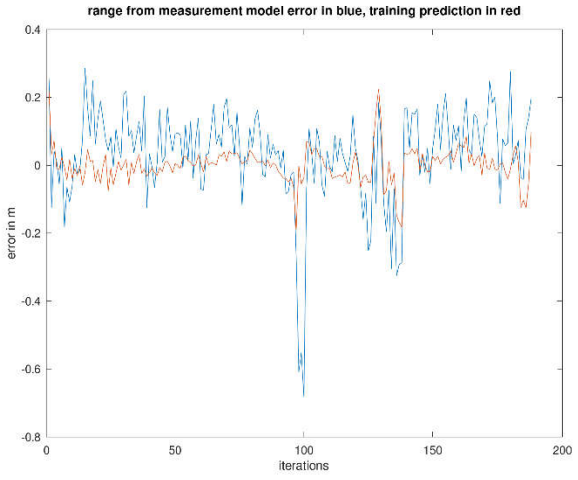
Figure 10. Zoomed in prediction views (GMM on left in dark blue, NN on right in faint blue)

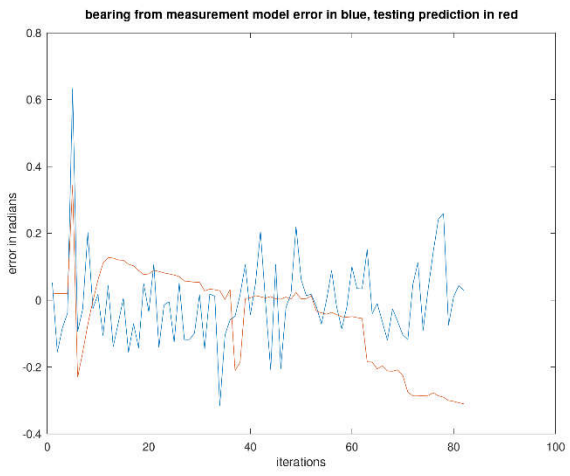
For the Range measurement we can see the R^2 is quite good for the training set and decent for the testing set. However, the error value is much higher compared to the training set. So even though the model has captured the 'variability' across the testing set, it still hasn't fit well.

For bearing the error is low for the testing data, implying good prediction, but the R^2 value is poor, showing it isn't capturing the distribution of the data. This again shows a sub-optimal fit. It also demonstrates the importance of not relying on a single evaluation metric to analyze the quality of the model.

These observations lead us to conclude that the neural network is able to generalize how well the robot moves with respect to the landmark, but has an error in accurately estimating its distance. Also, the NN manages to give a close estimate to how much the bearing value should be, but isn't able to pin-point its position in the 'circle' of possibilities (with a radius equal to the error value).

The aim of this ML process was to get a better model to predict sensor readings, than the analytical one at hand. To see whether this aim is achieved, range and bearing error from both the analytical measurement model described earlier and the NN predictions are shown below.





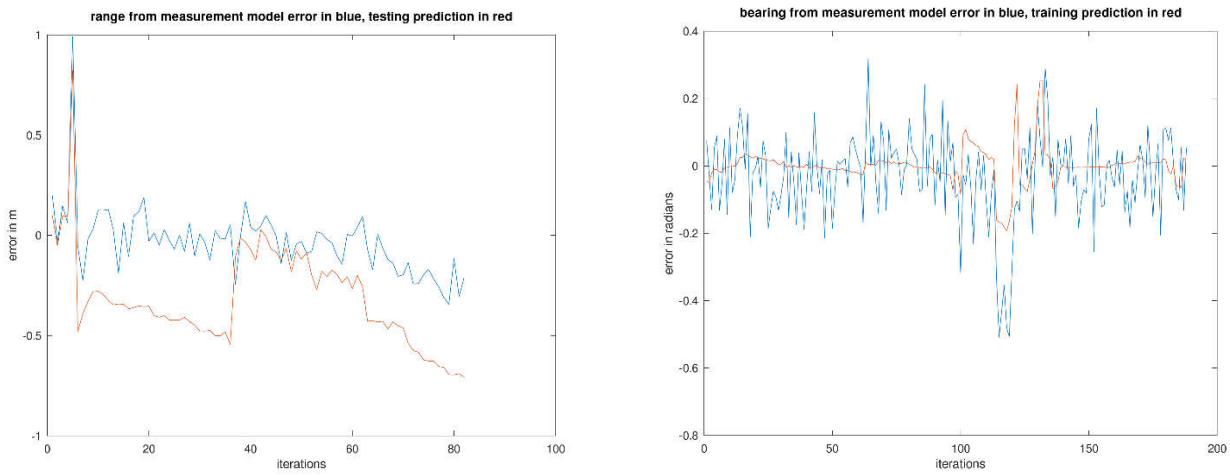


Figure 11. Range and Bearing error comparison for NN predictions (training on top/test bottom)

The same conclusion is formed looking at these plots. The NN for the training set predicts better than the analytical model, but does not perform better for either the range or bearing over the testing set. Clearly the GMM-GMR is better at predictions over the given data-set than the Neural Network. The difference in the prediction accuracy of two algorithms can be explained by the inherent trait of a NN. NN's are function approximators that are able to learn complex inter-relations in the input space. Even though they are quite powerful, they need a lot of data to form these relations. The dataset provided has only 288 training examples which are clearly not sufficient. In contrast the run from HW2 that learnt the motion model has similar or more complex features, but around 12000 training examples. Earlier metrics are 0.0015219 for the MSE and 0.99887 for the R^2, both on the testing set, which demonstrate quite good performance.

Question 5: Conclusion

1. GMM-GMR works better at predicting the learning aim than NN's
2. Gaussian Mixture models can find patterns in data that can enable them to be grouped together by a common thread. If this pattern finding is important, Neural nets as used here are not useful.
3. Since the GMM-GMR works well it can be concluded that there is a Gaussian nature to the input data.
4. It was slightly surprising that the output at-least for the training set was decent (especially range) even though the amount of data was not of a large magnitude as is expected for neural networks from literature.
5. Larger amount of training data might have led to better performance by the neural network.

Dean Pomerleau,(1992) "Neural Network Perception for mobile robot guidance", School of computer Science, Carnegie Mellon University.

Tom M Mitchell "Machine Learning"

Barnston, A., (1992). "Correspondence among the Correlation [root mean square error] and Heidke Verification Measures; Refinement of the Heidke Score." Notes and Correspondence, Climate Analysis Cente

"Anatomy and Physiology" by the US National Cancer Institute's Surveillance, Epidemiology and End Results (SEER) Program ., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1474927

LWLR Comparison – Michael Rencheck

Question 1: The New Learning Aim

The new learning aim to be applied to the locally weighted linear regression (LWLR) algorithm is a different dataset formulation of replacing the motion model. Instead of trying to predict the change in state given the controls and some aspects of the current state (the original learning aim), the goal of this new formulation is to predict the next state given the current state and the controls. Both datasets were based on dataset 1 from the UTIAS study[1]. Table 4 compares the differences between the two learning aims.

Although, the learning aims are attempting to effectively achieve the same goal of estimating the true motion model, the implementation varies in a non-trivial way. In addition to predicting a different output, the overall input space is larger than the original learning aim. In addition to that, angle wrapping of the groundtruth heading in the new learning aim was handled by calculating the sine and cosine of each value. Whereas the original learning aim implemented angle wrapping logic in the dataset formulation.

Table 4. Comparison of the old and new learning aim

Learn the change in state, $\Delta\vec{x}$		**Learn the next state, $\vec{x}_{t+1}$**	
Input	**Output**	**Input**	**Output**
v, ω, θ_t	$\Delta x/\Delta t$	$v, \omega, x_t, y_t, \sin(\theta_t), \cos(\theta_t)$	x_{t+1}
v, ω, θ_t	$\Delta y/\Delta t$	$v, \omega, x_t, y_t, \sin(\theta_t), \cos(\theta_t)$	y_{t+1}
v, ω, x_t	$\Delta\theta/\Delta t$	$v, \omega, x_t, y_t, \sin(\theta_t), \cos(\theta_t)$	$\sin(\theta_{t+1}), \cos(\theta_{t+1})$

There is also some difference in the transferability between scenarios. The original learning aim is trying to predict a relative change in state, so this training data could be transferable if the robot was operated differently under the same environmental conditions, especially the change in x and y predictions. For the new learning aim, the training data is more specific to the measurement set up. Therefore, this training data might need to be modified to be reused. For example, if the world coordinates change, the training data would need to be modified to match the new world frame. These differences resulted in very different models of data given the different input spaces. Figure 12 shows a graphical comparison for the x component of the expected output for each learning aim. The new dataset formulation results in a highly non-linear data model with minimal noise across the input space which is very well suited to the LWLR algorithm.

Question 2: Learning Algorithm Comparison

Initially this dataset was used in conjunction with a neural network, which in theory can be very different or very similar to a locally weighted regression. This implementation was a network that only consisted of one hidden layer using the sigmoid activation function, so there are a few key differences compared to the LWLR.

[1] (Halpern, Lui, Barfoot, & Leung, July 2011)

Both algorithms require a set of data to be used as the training set and a set of data to test how well the training data represents the learning aim the algorithm is trying to predict. While both algorithms are a form of supervised learning, meaning they require a known/expected output for each set of inputs, the first major difference is that a neural network must be trained prior to making a prediction. Depending on the size of the training set and size of the network, this can lead to long training times required before any prediction can be made. However, once the network is trained, each query is calculated very quickly even for a large test dataset since the network is essentially performing a series of multiplications and summation. LWLR is a form of lazy learning, meaning that prior to making a prediction, there is no training time required. Even though this method requires no training time, the actual time to make a prediction is directly correlated to the size of the training dataset and the number of inputs. In the final equation[2] to make a prediction for the LWLR, shown below, there is a matrix inversion.This portion of the computation will take the most time, especially for large training datasets. For the training datasetused for the new learning aim, a single prediction takes roughly half a second.

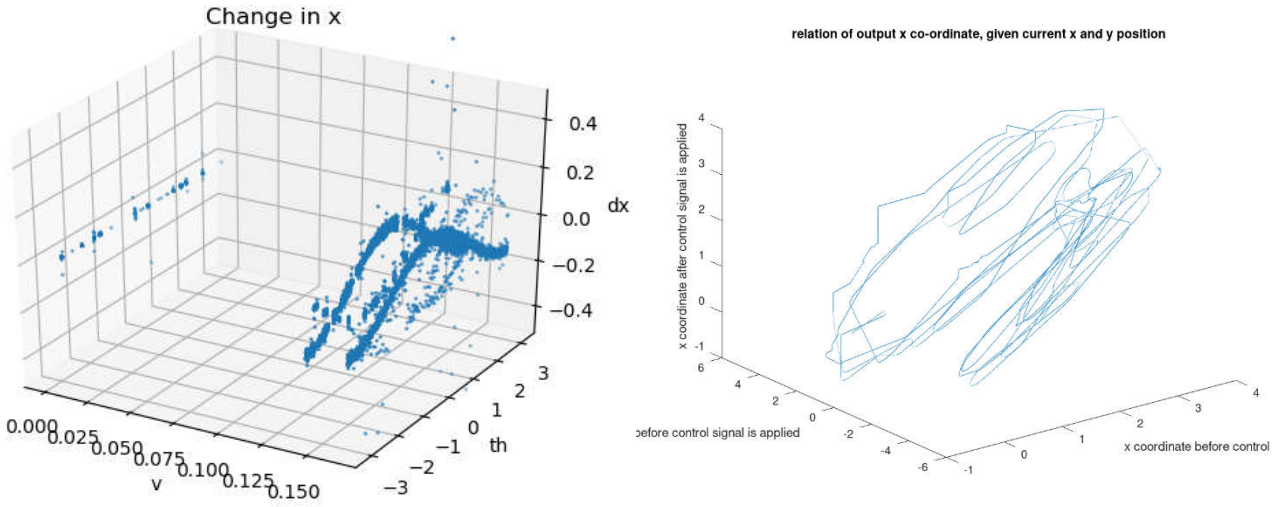


Figure 12. Comparison of the x output across some of the input space for each learning aim.

$$\hat{y} = \boldsymbol{q}^T(\boldsymbol{Z}^T\boldsymbol{Z})^{-1}\boldsymbol{Z}^T\boldsymbol{v}$$

q = query point
$\hat{y}$= the predicted output
Z = weighted training data input matrix
v = weighted training data output vector

Another difference is how each algorithm is using weighting factors. In LWLR, the weighting factors are calculated directly based on a distance function, relating how far the query point is from each point in the training dataset. This is then put through a kernel function in order to create an inverse relationship between the distance and the weight; as distance increases, the corresponding weight should decrease. These weighting factors are completely dependent on the query point and are recalculated each time a new query is made. For a neural network the weighting factors are determined through trial and error during the training phase. A set of weights are randomly initialized for every connection to each unit for all layers of the network. The training data inputs are then propagated forward through the network and an error is computed between the predicted output and the known output. This error is then used to modify each weighting value by back propagating a stochastic gradient descent. This process is then repeated until the resulting prediction error is under a set threshold. These weights are then used to predict an output for each query in the test set.

[2](Atkeson, Moore, & Schaal, 1997)

Another major difference in the two algorithms, is the robustness to noise. Since the neural network is iteratively refining the weights across the whole training set, if there are any outliers in the training dataset, the resulting weights are minimally impacted by them. In contrast, these will greatly impact a prediction made using the LWLR algorithm. If the inputs corresponding to the outlier are close to the query point, the weighting factor attributed to the outlier will also be high, which will cause local linear fit (and subsequently the prediction) to be very inaccurate.

Lastly, a neural network has the advantage of being able to extrapolate and make accurate predictions even if the query was not directly inside the bounds of the training data input space. This is advantageous because there is a robustness when attempting to use the learned network in a general application. However LWLR is unable to make accurate predictions if the query is too far outside of the input space. As a given query get farther outside of the input space captured by the training data, the distance will grow resulting in small weighting values. This will produce a poor locally linear fit and return a prediction will be very far off from reality. This limitation may limit the usefulness of running a LWLR in a general application unless the training data sufficiently covers the input space.

Question 3: Algorithm Implementation on New Aim

Since the new learning aim was similar in format to the old learning aim (a regression problem with multiple inputs and corresponding outputs), very little effort was required to adopt the dataset to work with the algorithm. For this implementation, each output variable was assigned its own iteration of the LWLR algorithm to circumvent minimizing the error across four outputs. This allowed each output prediction to be computed directly allowing for a faster overall computation time. The only modifications that were made was adjusting the bandwidth parameter to dial in the best fit for this new dataset. For LWLR, the bandwidth is a design parameter to control the calculated weighting factor for each data point in the training set. It is represented by h in the following equations:

$$w_i = \sqrt{K\left(\frac{d(x_i, q)}{h}\right)}$$

$$K = exp\left(-\frac{d(x_i, q)^2}{h}\right)$$

$K()$ = Kernel Function
$d()$ = Distance function
h = fixed global bandwidth

Use a bandwidth of $h > 1$, the weighting factor increases for points farther away from the query. This creates a smoothing effect across the predicted output. However, smoothing too much will produce a poor fit of the predicted output. Conversely, using a bandwidth of $h< 1$, will increase weight of the data points very close to the query, making the local linear model more dependent on training data points very close to the query. Decreasing$h$ too much will result in over fitting and the LWLR will fail at capturing the underlying function. The effect of the changing bandwidth on a generic exponential decay function can be visually observed in Figure 13.

The h value can only be tuned through testing and will very for each implementation. For the old learning aim and dataset, the best results were obtained with h = 0.06, but for the new learning aim and dataset, the best results came with h = 10.Figure 14 shows a graphical comparison of the variance across thex_{t+1} prediction for new learning aim testingdataset for$h$ = 1 and h = 10. The h = 10 plot has a lower average variance across the entire dataset meaning that the predictions are a better overall fit across the whole test set. These same trend of a lower average variance for h = 10 was also observed for the other three output variables.

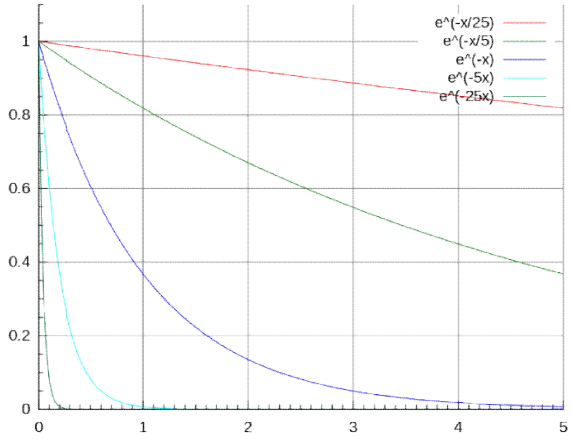


Figure 13. Graph of exponential decay functions similar to those used to calculate the weighting factors[3]

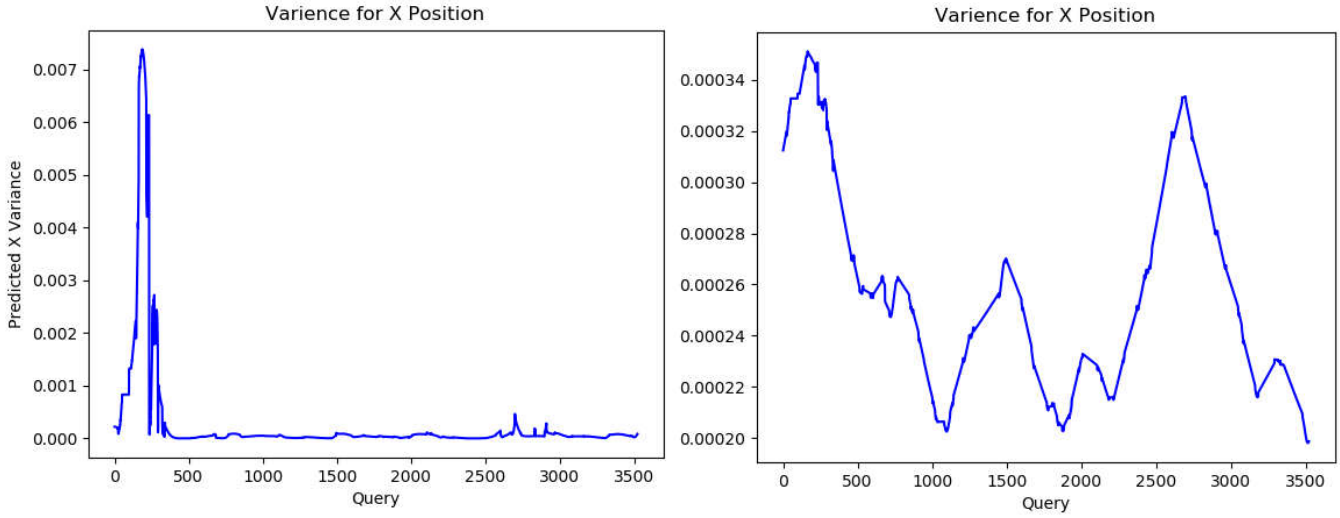


Figure 14. Variance comparison for *h*=1 (left) and *h*=10 (right)

Question 4: Assessment Comparison

The testing results in the figures below compare the performance of the LWLR algorithm to the neural network for the same training and testing dataset for the new learning aim. The total dataset had a 70/30 split between the amounts of training points to testing points. Also Table 5 shows a comparison of the calculated mean squared error across the testing set for each output variable. The MSE was calculated using the equation below:

$$MSE = \sum_{1=1}^{N} \frac{(\hat{y}_i - y_i)^2}{N}$$

Table 6 shows a comparison for the calculated coefficient of determination (R^2) across the testing set for each output variable. R^2 was calculated using the equations shown earlier.

The performance across both algorithms is good with the LWLR performing slightly better along some regions of the test set. Looking at the plot for the new dataset in Figure 12, it seems to imply that the output relative to the six dimensional input space is highly locally linear. Due to this fact, the local linear models fit to the dataset

[3](Autopilot, 2010)

extremely well producing a highly accurate prediction. The fact that LWLR prediction is made by only considering training data points that are close to the query point, helps explain why there is a slight improvement in the final prediction when compared to the neural network output. The prediction from the network is using weights calculated based on the entire training dataset which may be causing the slight error on some of the sharp corners.

These results are further reinforced by comparing the calculations of MSE and R^2 for both algorithms. The results for LWLR show a slight improvement over the values resulting from the neural network comparisons. It should be noted that the LWLR results for $\sin(\theta_{t+1})$ and $\cos(\theta_{t+1})$ are unrealistic and due to a numerical error in the computation. The variance plots for both output variables are non-zero across the dataset so there is definitely some error, however when calculating the numerator of the SS_{res} equation, the squared difference must result in a very small number such that the computation is being rounded to zero.

Since the learning aims were similar, an approximate comparison was also made between the old and new dataset. Since the datasets vary in the actual points used, this is not a direct comparison. However, both datasets were created from the same trajectory so it was possible to evaluate roughly the same region. These results are shown in Figure 18. This comparison demonstrates the importance of data set formulation before trying to run it through an algorithm.

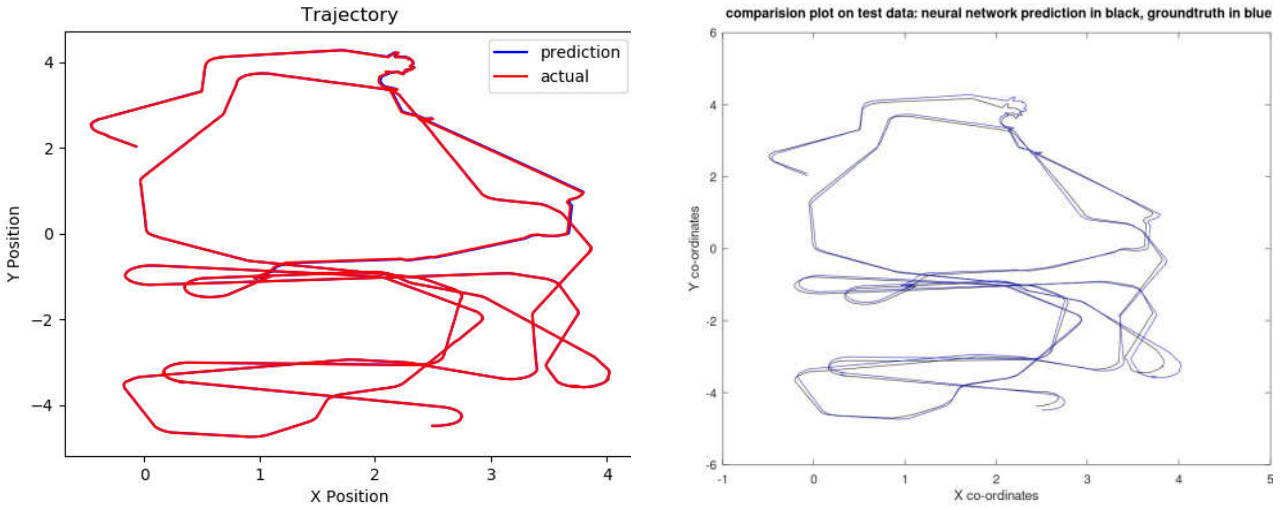


Figure 15. Predicted and expected trajectory comparison (left LWLR, right NN)

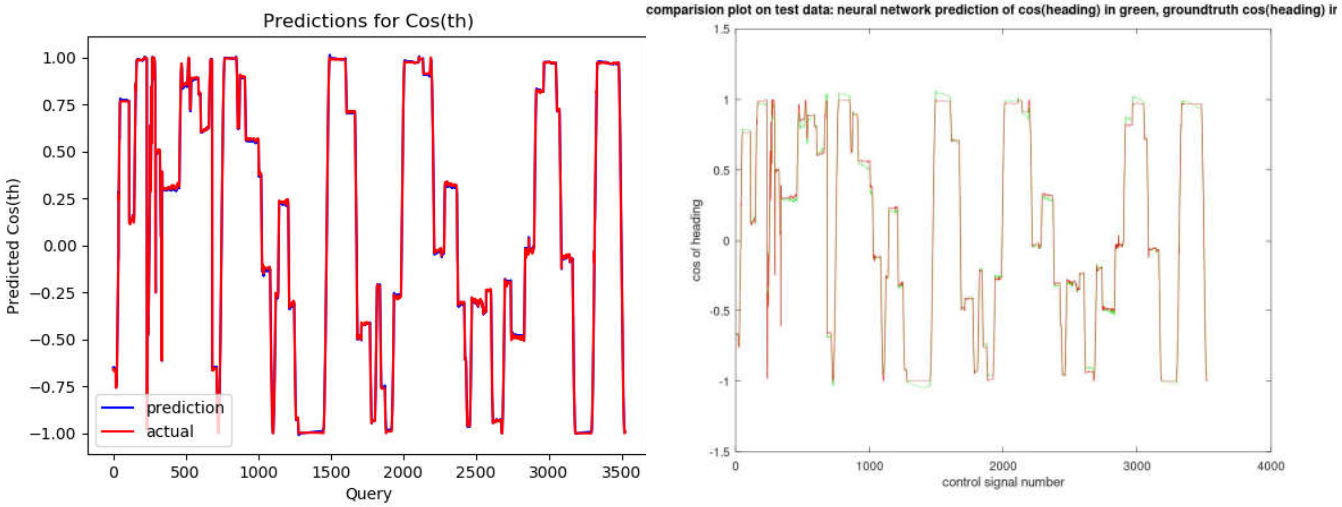


Figure 16. Predicted and expected *cos(θ)* comparison (left LWLR, right NN)

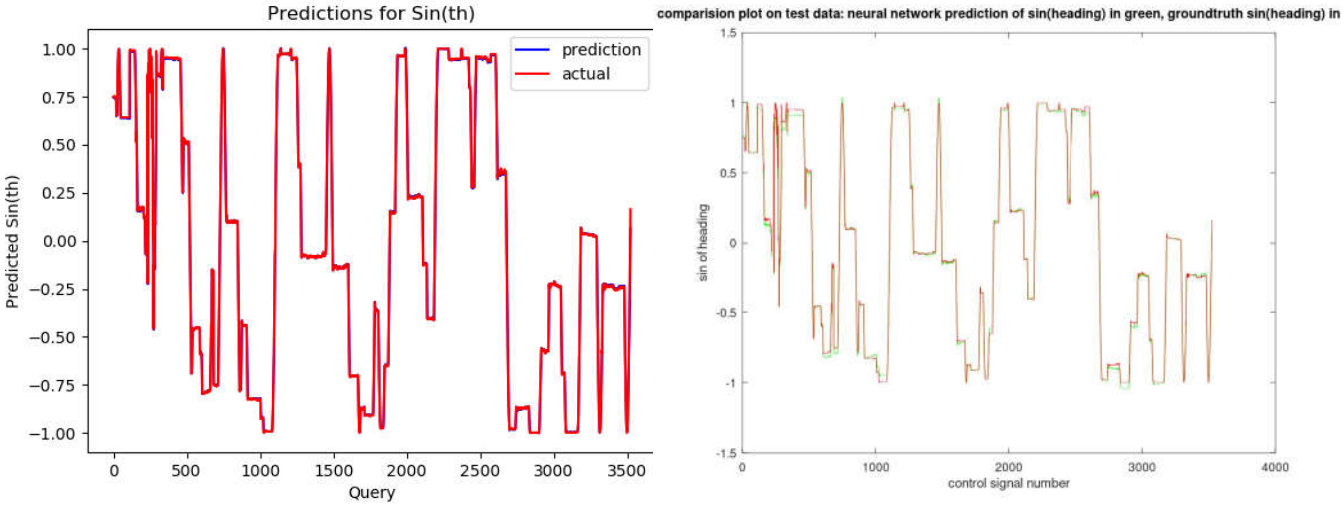


Figure 17. Predicted and expected *sin(θ)* comparison (left LWLR, right NN)

Table 5. MSE Comparison for testing data set on LWLR and neural network

	MSE x_{t+1}	MSE y_{t+1}	MSE $\sin(\theta_{t+1})$	MSE $\cos(\theta_{t+1})$
Neural Network	0.002545	0.0034131	0.0026013	0.0046155
LWLR	0.000337	0.0020741	0.0019773	0.0032508

Table 6. R^2Comparison for testing data set on LWLR and neural network

	R^2 x_{t+1}	R^2 y_{t+1}	R^2 $\sin(\theta_{t+1})$	R^2 $\cos(\theta_{t+1})$
Neural Network	0.99840	0.99941	0.99462	0.99077
LWLR	0.99978	0.99996	0.99591	0.99350

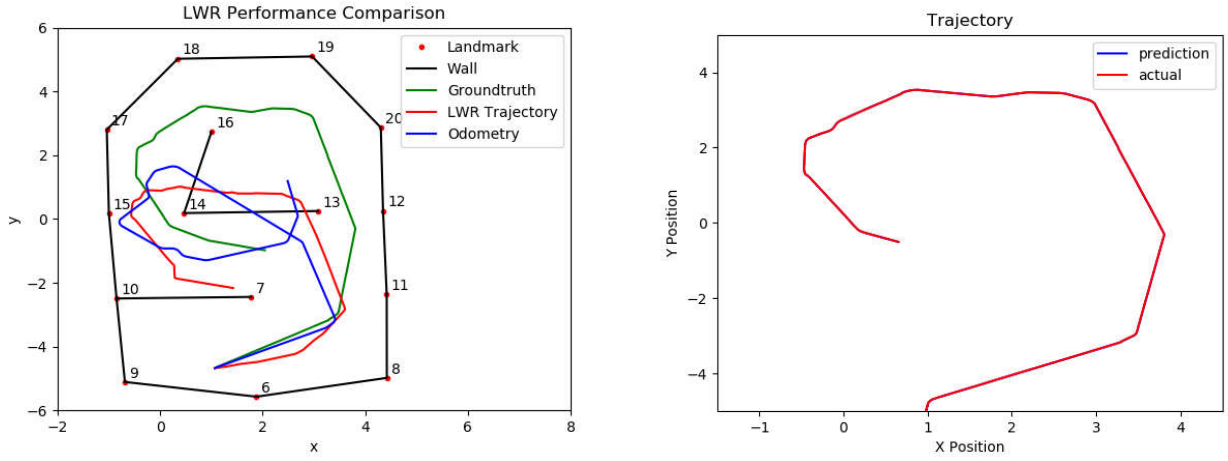


Figure 18. Comparison of the old and new dataset for the LWLR algorithm over roughly the same trajectory

Question 5: Conclusion

For this learning aim and dataset formulation both the LWLR and neural network algorithms perform very well with a slight edge given to LWLR. The LWLR performed slightly better to highly locally linear relationship between the inputs and various outputs, resulting in very accurate locally linear models of for each query point.

These results also demonstrate the variation by using a different dataset formulation. Even though the new and old learning aims were trying to achieve the same goal of replacing the motion model. The new dataset formulation preformed much better than the old dataset. This can be attributed to how the new dataset did a better job of handling angle wrapping as well as creating an input space more suitable to the LWLR algorithm.

The end goal of this experiment would be to use the learning algorithm in place of a filter in order to predict the true state of the robot. While the performance of the LWLR excels for the given training and test set, it may perform poorly if the robot travels into a new area of the input space in a future run. Since the algorithm cannot make a accurate predictions if the query is not captured by the training data, there is a risk that the prediction error will increase greatly and cause unpredictable behavior by the controller. Since the robot is operating in a relatively small area for dataset 1, this may not be an issue and it is also feasible to harvest more data to better cover the input space. However, for a larger or continuous environment the neural network would be the preferred option of the two.

Bibliography

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally Weighted Learning. Kluwer Academic Publishers.

Autopilot. (2010, July 30). *Exponetial Decay*. Retrieved from Wikipedia: https://commons.wikimedia.org/wiki/File:Plot-exponential-decay.svg

Halpern, Y., Lui, H., Barfoot, T., & Leung, K. (July 2011). The UTIAS Multi-Robot Cooperative Localization and Mapping Dataset. *International Journal of Robotics Research*, 30(8):969–974.