Machine Learning & Artificial Intelligence for Robotics - Homework 2

Locally Weighted Linear Regression for Motion Model Estimation

November 18th, 2019

Maurice Rahme

1 Part A: Learning Aim and Algorithm

1.1 Learning Aim

The aim of this assignment is to improve the Dead Reckoning error of the motion model from homework 0. Using the motion model, stated below, the ground truth path is not well predicted, especially in the presence of simultaneous linear and angular velocity commands, v, and ω respectively.

The subject evaluated here is a mobile robot moving on a 2D plane. Its equations of motion assume a unicycle model whereby motion can be achieved by a combination of linear and angular velocity, thus mapping the robot as a 3 degree of freedom system represented by coordinates x, y, and θ. Hence, the nonlinear motion model is the following for each iteration of time elapsed dt [1]:

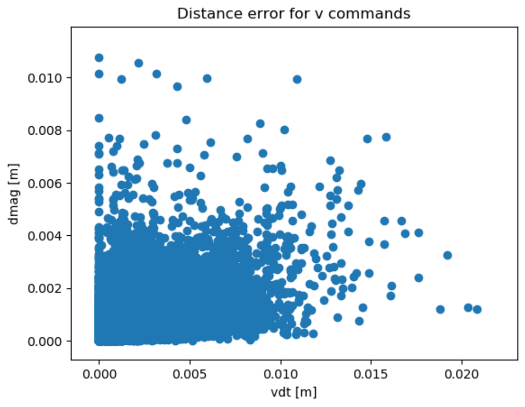
$$x_t = x_{t-1} + v * cos(\theta_{t-1}) * dt \tag{1.1}$$

$$y_t + = y_{t-1} + v * sin(\theta_{t-1}) * dt \tag{1.2}$$

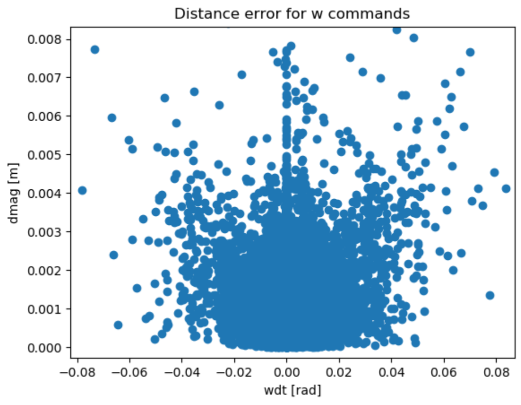
$$\theta_t + = \theta_{t-1} + \omega * dt \tag{1.3}$$

where v is linear velocity, and ω is angular velocity. The model is nonlinear due to the sin and cos relationship between the robot coordinates and velocity [1].

Below is the plot for position magnitude error against odometry commands.



(a) Position Error wrt v*dt



(b) Position Error wrt ωdt

Figure 1: Position Error for Odometry Commands times command duration

To produce this plot, the Odometry and Ground Truth data sets of around 96,000 entries were cleaned up using the following criterion: if a single odometry command falls between two ground truth entries on data set 0, then the row is saved, otherwise, if two or more odometry commands fall between any two ground truth entries, the row is discarded. This was done to eliminate the ambiguity of the effect of compounded odometry commands on change in position and heading. The resultant data sets, named 'gt_train.csv' and 'odom_train.csv' each contain approximately 56,000 entries. However, not all of these are used in training due to high computational cost. Additionally, the data sets contain many outliers, which, after being eliminated to produce the plots above, reduced the size of the training set of 42,666 elements. The figure above maps the difference between another data set 'gt_deadreck.csv' and 'gt_train.csv', whereby the former was created by applying the odometry commands in 'odom_train.csv' to each coordinate in 'gt_train.csv' to compute the change in position from the same starting point according to pure dead reckoning. The recorded variance on positional error and heading was calculated to be 6.36e-7 and 3.47e-5 respectively.

1.2 Learning Algorithm

For this task, Locally Weighted Linear Regression (LWLR) was deemed a good solution as it can fit nonlinear input-output relationships thanks to its tunable parameters; the number of training data points (inputs and outputs), as well as the bandwidth k, which defines the range over which a fit is produced. LWLR is a lazy learning method as training is done in real-time at the evaluation of each query point, the downside being that it is slow to run, and the upside being that it requires limited memory. Linear Regression usually under-fits data, especially nonlinear data, as it returns the lowest mean-squared error for its un-biased estimate of each point in the data set. LWLR overcomes this limitation by assigning higher weights to data points the closer they are to a query point x_q as defined by a Kernel function [2].

As this learning algorithm uses local models for training, it is helpful to keep these simple by assuming linearity in the locally evaluated range. Hence, at each local space, the models are said to be linear in their unknown parameters, which are evaluated using the least squares training criterion. Here, the input data for training and testing is a two-column matrix of vdt and ωdt values in each row, and output data for training and testing is a two-column matrix of d and h values, where the former is change in distance magnitude and the latter is change in heading [2]. Before deriving the LWLR formulation, a Global Linear Regression Model is first presented:

$$X\beta = Y \tag{1.4}$$

Where X is the training input set of dimension $n \times d + 1$, where n is the number of row entries and d is the dimentionality of X_i for the i^{th} row of set X with a column of ones appended to the set to include a constant term in the regression. Similarly, the training output set Y has dimension $n \times p$, where p is the dimentionality of Y_i. Following from this, it is possible to solve for β, the solution to the Global Linear Model fit [2]:

$$\beta = (X^T X)^{-1} X^T Y \tag{1.5}$$

Where β can be multiplied by any query point x_q to achieve a predicted estimate output $\hat{y}$, which can be compared agaist an entry Y_i for performance evaluation. Following from this, the LWLR algorithm can be derived, whereby each local model is built upon the formulation described above [2].

For each query point x_q in the input testing data set, a distance is calculated between x_q and each input X_i of the training data set X. A Global Euclidean distance $d_e = \sqrt{(x_i - x_q)^T(x_i - x_q)}$ is used in this case, although there is the option of using a different distance calculation for each x_q. This Euclidean distance is then fed through a Gaussian Kernel function after being divided by the bandwidth parameter k described above. The Kernel is calculated as

$$K = e^{(d_e/h)^T(d/h)} \tag{1.6}$$

and is fed to the diagonal weight matrix W entry $W[i,i]$ for each input x_i of the training data set X as

$$W[i,i] = \sqrt{K} \tag{1.7}$$

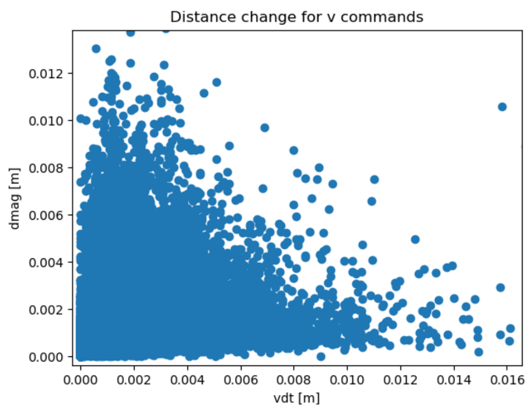
From here, each row i of the training data set's inputs X and outputs Y is multiplied by the weight matrix W to create the variables Z and V respectively, where $Z = WX$ and $V = WY$ [2].

Here, β is solved for each x_q using these new variables:

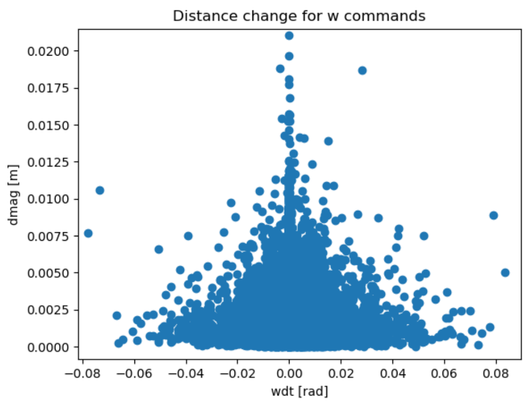
$$\beta = (Z^TZ)^{-1}Z^TV \tag{1.8}$$

where the output corresponding to each x_q is found using $\hat{y}(q) = x_q^T\beta$ [2].

To justify the use of LWLR in this task, local linearities were identified in relationships between positional change, heading change and odometry commands. Here are the positional change plots for vdt and ωdt commands times their duration with removed outliers:



(a) Position Change wrt v*dt

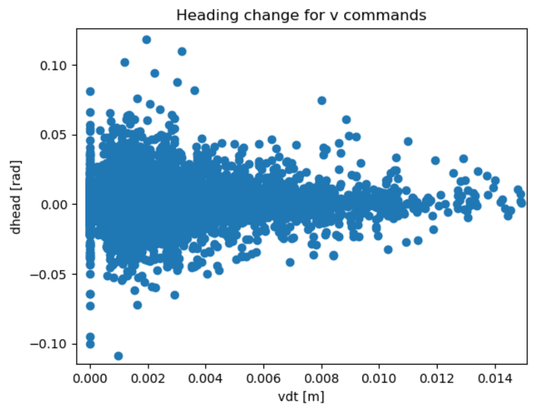


(b) Position Change wrt ωdt

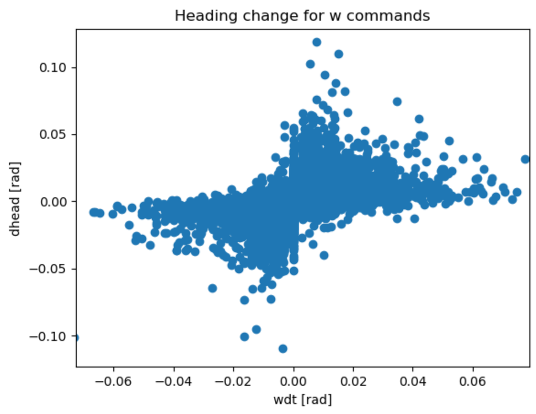
Figure 2: Position Change for Odometry Commands times command duration

As expected, the greatest positional change in ground truth data occurs for ω commands approaching zero, showing locally linear behaviour between $= \pm 0.06$ and 0 wherein the exponential growth towards ωdt leaning to zero can be approximated by local linearity. Interestingly, however, the relationship between positional change and vdt is not as straightforward, and instead seems to exhibit an exponential decay curve, likely due to its coupled relationship with simultaneous ωdt commands.

Note that vdt and ωdt were chosen instead of ω and v as these provide a more accurate representation of the nature of the odometry commands by encompassing the duration of the applied commands. Here are the heading change plots for vdt and ωdt commands times their duration with removed outliers:



(a) Heading Change wrt v*dt



(b) Heading Change wrt ωdt

Figure 3: Heading Change for Odometry Commands times command duration

Here, the greatest heading change seems to occur at vdt commands nearing zero, perhaps speaking to the properties of the robots' wheels, which may slip and spin in place at higher speeds, resulting in less movement than anticipated. Interestingly, the heading change appears to fan out as ωdt approaches zero, and to decay exponentially as ωdt approaches ± 0.06.

1.3 Testing the Learning Algorithm

A noisy sine wave was produced in code, and a LWLR using the Gaussian kernel was deployed to successfully fit an approximation on new data. This was done by changing the offset of the sine wave's first input element (time) as well as the time step between elements and the number of elements (200 training points and 300 testing points), for which the plot can be seen below. LWLR works well for univariate regression, but it will face a bigger challenge when dealing with multivariate regression [2].

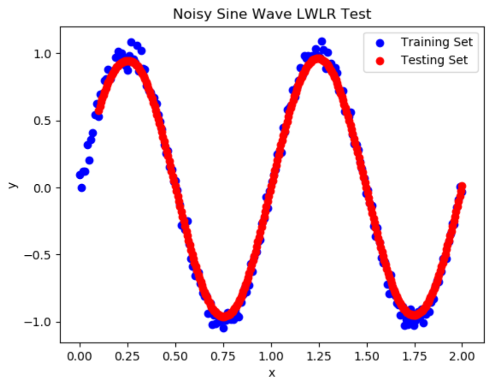


Figure 4: Testing LWLR on a noisy sine wave.

2 Part B: Performance

2.1 Leave One Out Cross Validation

Cross-validation assesses how well LWLR (or any Machine Learning Algorithm) performs against unseen data. Although a straightforward performance measure is the difference between $\hat{y}_i$ and x_i, this value will be deceptively low for bandwidths which cause overfitting. With Leave-One-Out-Cross-Validation (LOOCV), the i_{th} prediction error is found by setting the training input set as the testing input set, and for every x_q, removing the point from the training set during evaluation. As a concrete example, if there are 100 elements in set A, LOOCV will evaluate A_0 against $A_1 : A_{100}$, then A_1 against $A_0, A_2 : A_{100}$, etc. With this lazy learning method, where the computational cost is concentrated at prediction time, it takes the same amount of time to perform a prediction with one less element as it would with the element included [2]. Hence, the local mean squared error (MSE) for a single query point x_q is

$$MSE_q = \frac{1}{n_{LWLR}} \sum_i \frac{r_i}{1 - Z_i^T (Z^T Z + \Lambda)^{-1} Z_i)} \tag{2.1}$$

Here, Λ is a diagonal matrix with small positive elements λ_i^2 (here set to zero), which serve to add 'fake data' to the weight matrix in a method known as Ridge Regression if there are not enough nearby non-zero-weighted points x_i for a given query point x_q, causing there to be not enough equations to solve for β and resulting in a non-invertible matrix $Z^T Z$. In practice, this is circumvented by using a pseudo-inverse for $(Z^T Z)^{-1}$, hence why the elements of Λ are set to zero [2].

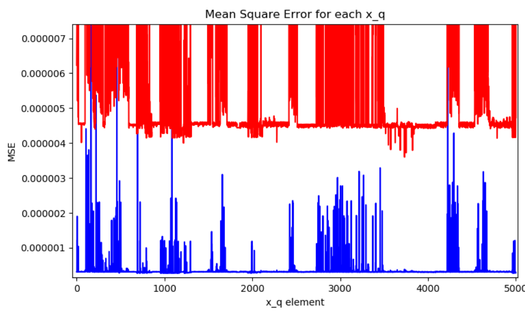
Additionally, r_i is the weighted residual (error) at all training points when evaluating x_q, and is given by $r_i = Z_i^T \beta - V_i$. Finally, n_{LWLR} is a measure of the actual number of data points, calculated as $n_{LWLR} = \sum_{i=1}^{n} w_i$ for x_q [2].

Following from this, the local noise variance for each x_q is

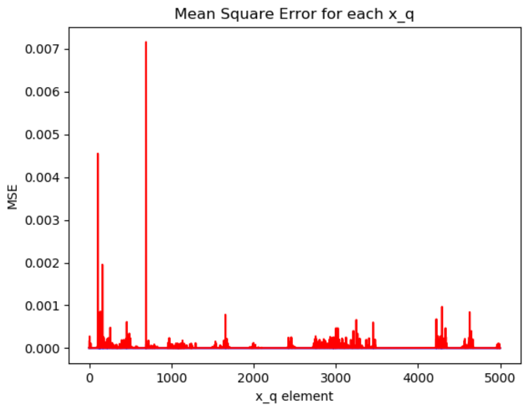
$$\hat{\sigma}^2 = \frac{\sum_i r_i}{n_{LWLR}} \tag{2.2}$$

2.1.1 Mean Squared Error and Tuning

A good way to tune the bandwidth parameter k (and a good measure of performance) is to run LOOCV on a significant and representative portion of the data set (5000 points seems to cover the whole range of input-output values according to the plots, and any more is infeasible to run) and evaluate performance for a particular k value by summing the MSE for each x_q, whereby k should be chosen to minimise this sum [2]. The issue with this tuning is that there is a relationship between k and the number of elements in the training set, and increasing the size of the training set increases computational cost. As a result, there is certainly room for further tuning for these two parameters. Below is a plot for Mean Squared error across all x_q denoted by their position in the training set for $h = 8e - 5$, with a sum of 0.0106 meters for distance magnitude change and 0.4410 radians for heading change.



(a) Distance MSE using LOOCV



(b) Heading MSE using LOOCV

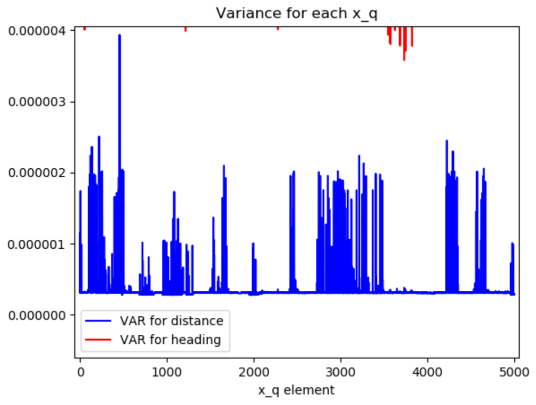
Figure 5: Mean Squared Error for each element x$_q$

Note that the data in blue shows MSE for distance and the data in red shows MSE for heading.

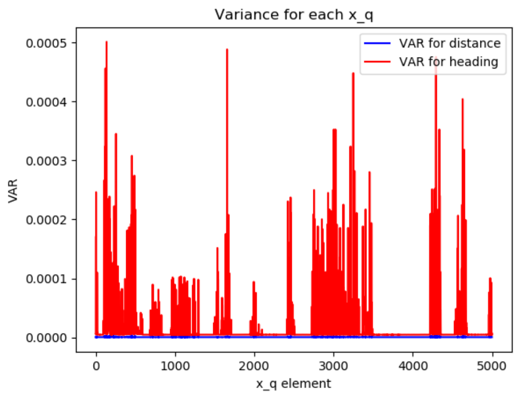
This configuration of k, the kernel choice and the number of training points yielded the lowest MSE for distance and heading of the trialled options. This is not to say that the configuration is optimal, as an optimization was not performed explicitly, as is recommended in the Global Tuning method [2]. However, examining the MSE for heading, there are significant spikes, perhaps suggesting that a global parameter fit is not suitable for this application, and that query-based local tuning, whereby the parameter fit is optimized for each query point, may be a better option, albeit with an even higher computational cost. Additionally, the kernel choice is generally not as critical, but other options could be explored for further optimization [2].

2.1.2 Variance

Another good measure of performance is variance, or the output data spread. Below is a plot for Variance across all x_q denoted by their position in the training set for $h = 8e - 5$, with a sum of 0.00208 meters for distance magnitude and 0.08413 radians for heading change.



(a) Distance Variance using LOOCV



(b) Heading Variance using LOOCV

Figure 6: Variance for each element x$_q$

From this figure, it seems that the variance for distance and heading are low and in the same range as was seen for the position and heading error plots in section 1.1.

2.2 Improving Dead Reckoning Error

The LWLR-estimated motion model performed better than pure dead reckoning for a bandwidth k of 8e-5 as shown in the plot below.

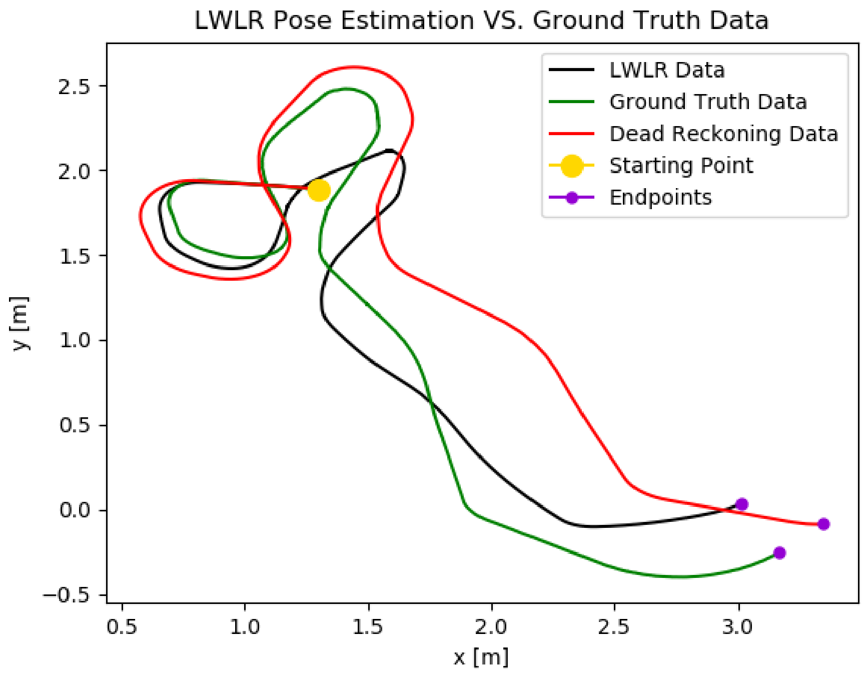
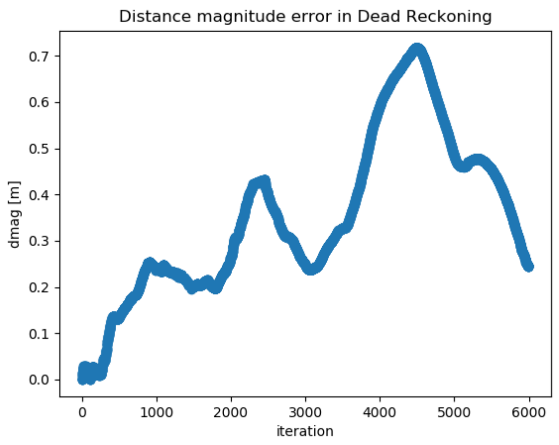


Figure 7: Dead Reckoning, Ground Truth, and LWLR-fitted paths for 6,000 iterations.

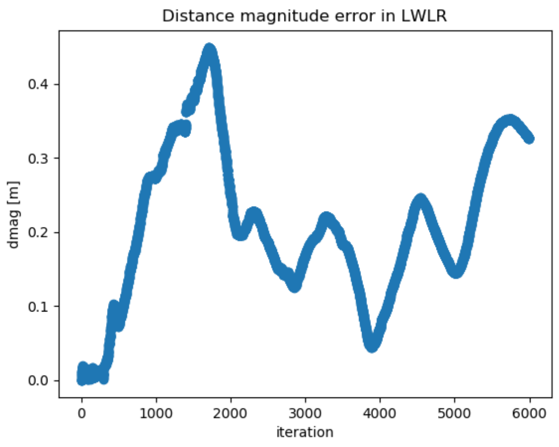
Here, it seems that the LWLR paths hugs the ground truth path closely on the first corner, but under-fits the second corner, creating an initial offset. This offset will never be corrected, even if future path updates are accurate, leading to persistent error for the rest of the trajectory. This indicates that more points needed to be sampled during the second corner. Given that the training data set contains 42,666 elements, it was not possible to use the full training data with reasonable execution times, which explains the relatively small 5,000-element training set size. However, one way to work around this limitation would have been to sample data points randomly instead of sequentially using the first 5,000. Aside from the somewhat poorer cornering behavior on the first two turns, the third corner with a positive ωdt command seems to have been followed accurately according to the shape of the curve. This indicates that a larger sample of negative ωdt commands would have served the fit better, meaning that if one were to sample the training data set, it would be beneficial to bias the sampling for more negative ωdt commands to better represent the model's behaviour for those inputs.

Additionally, at around position (1.6, 2.2), there is a small jagged change in motion, indicating over-fitting at this section, and further suggesting the use of query-based tuning for bandwidth k.

For a better look at LWLR performance, the below figures show the distance magnitude error for the dead reckoning and LWLR paths, measured by taking the distance magnitude between the ground truth position and the predicted positions at each iteration.



(a) Distance Magnitude Error in Dead Reckoning Path.



(b) Distance Magnitude Error in LWLR Path.

Figure 8: Distance Magnitude Error against Ground Truth for 6,000 iterations.

Here, it is evident that the dead reckoning path exhibits nearly twice the distance magnitude error than that of the LWLR path. In fact, taking the means of these plots, the average dead reckoning positional error was 0.3507 meters, and the average LWLR positional error was 0.2106 meters for 6,000 steps. Also note that the axes are not equivalent for both plots, so although they appear similar at first glance, the dead reckoning positonal error is consistently higher for most iterations.

2.3 Qualitative Performance Discussion & Improvements

The LWLR's performance in improving dead reckoning error was good, averaging a 40% reduction in positional error. Although it performed better than pure dead reckoning, averaging a position magnitude difference of 0.2106 meters for 6,000 odometry iterations compared to pure dead reckoning at 0.3507 meters, it is difficult to say whether this improvement is maintained over the whole data set, as the relationship between the training set size and the bandwidth k played a large role in defining performance, particularly during segments with nonlinear motion.

Additionally, the training data itself has a flaw wherein it is assumed that for each ground truth pose in 'gt_train.csv', the robot's velocities prior to the odometry command are 0, and that the command's duration is equal to the time of application to the time of the next recorded ground truth pose. A better data set would compound all previously applied velocity commands and subsume them into each entry of the training set 'odom_train.csv'. Note, however, that the training data modifies the entries in 'odom_train.csv' to instead use ωdt and vdt as its inputs as a way of creating an input training set which better represents the effect of odometry commands on position.

Additionally, the decision was made to define the training outputs as change in distance magnitude and heading. Another viable option would be to replace the change in distance magnitude with change in x and y position, resulting in a three-dimensional output. The code was written in such a way that this is easily modifiable, by swapping out one variable, but trials did not indicate better performance, although not much time was spent on this alternative.

Finally, the design choice was made to use direct data weighting instead of searching for a β which minimizes training error for each x_q, as this is computationally more efficient. The drawback of this is that it inherently causes interference for couples outputs, which may be a source of persistent error in this implementation.

3 Citations

[1] “Robot Motion.” Probabilistic Robotics, by Sebastian Thrun et al., MIT Press, 2010.

[2] ”Locally Weighted Learning”, Artificial Intelligence Review, by Christopher G. Atkenson et al., Kluwer Academic Publishers, 1997.