ME 469: Learning - LWR

MICHAEL RENCHECK

Question 1: Building a Data Set

A robotics study was conducted at the Autonomous Space Robotics Lab at the University of Toronto Institute for Aerospace Studies (UTIAS) to use multiple robots for Cooperative Localization and Mapping. In conjunction with a paper, the full data set for the robots used in the experiment was also published.[1] For this report, the files in dataset 1 were used to identify a possible learning application. The files include linear and angular velocity control commands, the ground truth position of the robot, measurement readings to reference locations from the onboard camera, and the groundtruth locations of the reference objects.

One of the biggest deficiencies with most mobile robotics platforms is the inability to determine an accurate believed state based solely on velocity commands and a kinematic model. For a differential drive robot, such as the ones used in the study, the kinematics can be calculated with the following equations:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} -\frac{v}{\omega}\, sin\theta_{t-1} + \frac{v}{\omega} sin(\theta_{t-1} + \omega\Delta t) + \varepsilon_x \\ \frac{v}{\omega}\, cos\theta_{t-1} + \frac{v}{\omega} cos(\theta_{t-1} + \omega\Delta t) + \varepsilon_y \\ \omega\Delta t + \varepsilon_\theta \end{bmatrix} \qquad \textbf{(1)}$$

Where:
v = linear velocity (input)
ω = angular velocity (input)
x, y = coordinate positions that define the robots location
θ = angle the robot is facing relative to the global positive x-axis
ε = random value from a Gaussian noise distribution
If *ω* = 0:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} -v\, sin\theta_{t-1} + \varepsilon_x \\ v\, cos\theta_{t-1} + \varepsilon_y \\ 0 \end{bmatrix} \qquad \textbf{(2)}$$

The inaccuracy of this method is due to the fact that these equations do not model key, real-world phenomenon, such as slip, that drastically impact a state calculation. It is also possible to try and model the entire system dynamically, but often this is not possible due to complexity and localized variations in the real world objects. For example, on a four wheeled robot, each wheel has the same general properties, but one wheel might have a small area of tread that was manufactured poorly. This will lead to worse traction and, over time, cause an accumulation of error between a simulation and the robot's physical world performance. However, instead of trying to model the physics behind every event, one could attempt to predict an outcome based on previously observed behavior. This report is an investigation into the ability to predict a change in state using the provided groundtruth data and odometry data.

The ideal goal would be to develop a training model that would rely on inputs independent of a given environment. This would then allow the robot to be placed in any similar environment and the same training set could still be able to make an accurate prediction. To create this model from the given dataset the following general process was followed:

1. Determine the state of the robot when each velocity command was issued.
2. Calculate the change in the state resulting from each command.

The VIACOM camera system recording the ground truth positions of the robot did not take readings that were synchronized with the issued velocity commands. In fact, for this dataset, there were multiple groundtruth positions recorded in-between consecutive velocity commands. So to achieve step one, the state matched to each velocity command was assumed to be the first groundtruth position recorded after the timestamp of the velocity command.

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

This assumption will create some small inaccuracies in the results of step two if the match position was recorded significantly later than the control was issued. In this data set, there were sections of the groundtruth data where no data was recorded for multiple seconds, resulting in some velocity commands matched with a state recorded seconds after it was issued. In order to reduce this source of error, the time stamp of the velocity command was compared to the time stamp of the matched groundtruth position and if the difference was greater than 0.05 sec, that data point was removed from consideration.

Next, the change in state that resulted from each velocity command was calculated from the matches made above. However, since some data points were removed from consideration, this created gaps where it wouldn't make sense to calculate the change in state. So when these gaps occurred, a change in state was not calculated. Then any points without a calculated state change were discarded from the dataset. Also, in the data, θ is always mapped to a range of $[-\pi, \pi]$ so the change in theta calculation was also mapped to handle this property.

Lastly, in order to account for the varying time steps, the change in state calculation was normalized to one second by dividing by the change in time. A variable representation of the dataset is shown below in Table 1.

Table 1. Dataset represented using variables.

Time Step (s)	***v* (m/s)**	***ω* (rad/s)**	***x* (m)**	***y* (m)**	***θ* (rad)**	***dt (s)***	***dx (m)***	***dy (m)***	***dθ (rad)***
t_i	v_i	ω_i	x_i	y_i	θ_i	$t_{i+1} - t_i$	$(x_{i+1} - x_i) / dt_i$	$(y_{i+1} - y_i) / dt_i$	$(\theta_{i+1} - \theta_i) / dt_i$
t_{i+1}	v_{i+1}	ω_{i+1}	x_{i+1}	y_{i+1}	θ_{i+1}	$t_{i+2} - t_{i+1}$	$(x_{i+2} - x_{i+1}) / dt_{i+1}$	$(y_{i+2} - y_{i+1}) / dt_{i+1}$	$(\theta_{i+2} - \theta_{i+1}) / dt_{i+1}$

Now that the dataset was assembled, various plots were made to identify useful or unexpected relationships, in order to guide the input selection. Figure 1 and Figure 2 show some of the plots used to identify potential sets of input variables. From kinematic equations above, it is clear that at a minimum the inputs and output should look like the following:

- v, ω, θ, and dt should predict dx
- v, ω, θ, and dt should predict dy
- ω and dt should predict $d\theta$

However since the state changes were normalized to one second, the time change component has already been accounted for in the data.

Figure 1 shows the linear and angular velocity command plotted against the change in each state variable. Figure 2 shows the linear velocity command and the current heading plotted against the change in x and y as well as ω and x against the change in θ.

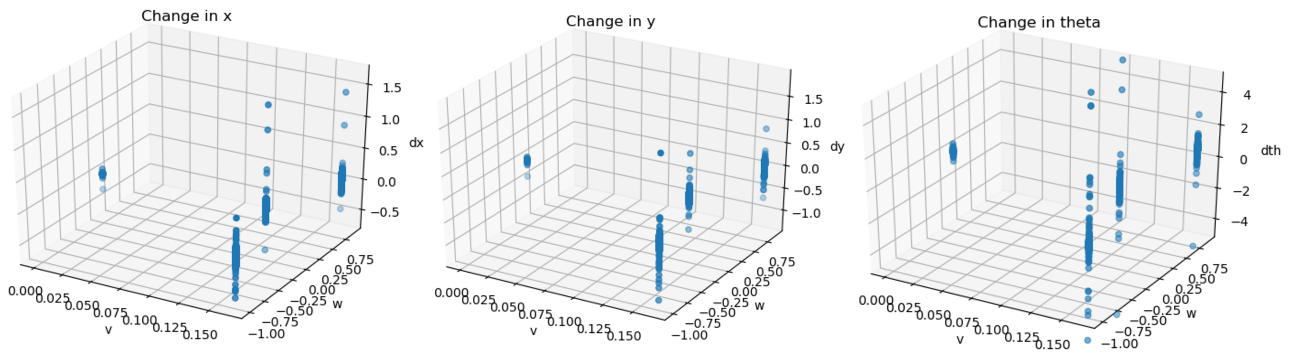

Figure 1. Linear and angular velocity command plotted against the change in each state variable.

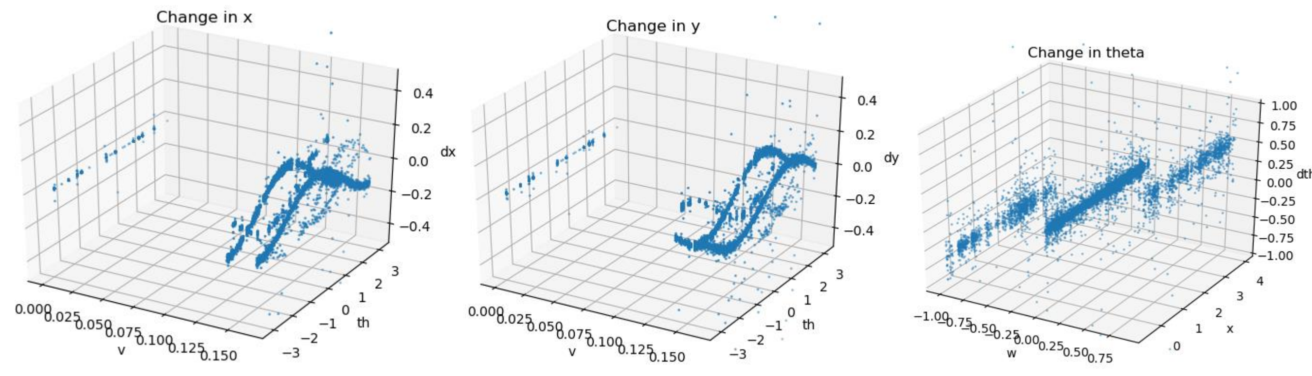

Figure 2. Velocity command and the current state plotted against the change in each state variable.

When analyzing the plots in Figure 1, the first interesting thing to note it that for this dataset, there are basically only 4 unique for velocity commands: stopped, straight, left turn, and right turn. As seen in the plots, this produces something akin to a distribution for each combination of *v* and *ω*. When both velocities are zero, the change in state is roughly zero and the variation shown in the plot can be attributed to the noise in the groundtruth camera measurements. This is an expected behavior and helps validate that the data has been plotted correctly. One other interesting note is that, one would expect the change in heading to be zero when the angular velocity command is zero, but the plotted data seems to contradict this thought. So even though the kinematics would imply the change in heading is only determined by *ω*, our training model should also include *v*.

Looking at the plots in Figure 2, the first observable relationship is the expected one: for *dx*, sin(*θ*) shows up and for *dy*, cos(*θ*) shows up. Since the robot was never issued a negative velocity command, it is expected that the change in *x*, *y* position should be dependent on the heading of the robot. However, these plots also reveal that there are recorded data points that seem to contradict this relationship. These deviations are highlighted in the plots shown in Figure 3. Also looking at the change in *θ* plot, we see the expected relationship that the change in heading has no dependence on the current *x* position (*y* was also plotted and had no relationship). However, it is observable that when *ω* =0, the heading has non-negligible variations ranging from ±.25 rad as noted in Figure 1. Also, since only using *v* and *ω*, seems to provide a distribution of data, *x* was also used as an input for the change in heading prediction to create a data model with locally linear relationship.

Based on these findings, it was determined that there was potentially some locally linear relationship between the state change and the inputs. So this study decided to explore the use of Locally Weighted Regression to predict the change in state.

Question 2: Algorithm Implementation – Locally Weighted Regression

Based on the resulting plots, there seems to be some form of locally linear relationship across some of the inputs, so in order to attempt to learn this function, an implementation of Locally Weighted Regression created to analyze the dataset created above. All the details discussed in this section were informed by the (Atkeson, Moore, & Schaal, 1997) paper on Locally Weighted Learning.

In general, regression attempts to create a single function that best describes a given set of data. Performing unweighted regression will yield a linear fit ($y = mx + b$) for all of the data points because each point is defined as having equal relevance. But the key difference in locally weighted regression is that each point is not defined as having equal relevance, but instead the relevance is determined based on a query given to the model. The equations below show the unweighted regression equation and the modifications for the locally weighted version.

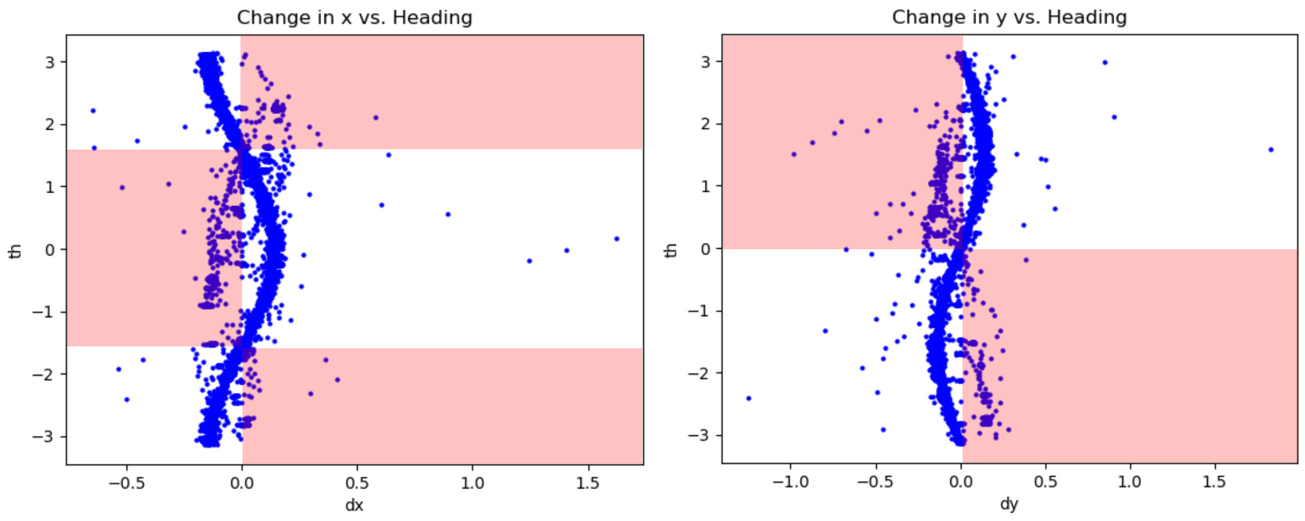

Figure 3. Current heading plotted against the change in x and y. The points inside the red regions are not expected with intuition, but are never the less represented in the data.

Unweighted Regression:

$$\boldsymbol{X}\beta = \boldsymbol{y} \qquad (3)$$

X = N x (n+1) sized matrix. N is the number of data points and n is the number of inputs to perform the regression. The "+1" is column of 1's to account for the constant term in linear regression.
β = (n+1) x 1 sized matrix. This can be thought of containing all the m and b terms for each data point that describe the linear regression.
y = N x 1 matrix. This is the output corresponding to a row of inputs x_i inside of X

Locally Weighted Regression:

$$\boldsymbol{wX}\beta = \boldsymbol{wy} \qquad (4)$$

W = N x N sized matrix. Each diagonal term, w_i corresponds to the weight of a given data point. All other terms not on the diagonal are 0.

In order to determine each diagonal term, the following equations are used:

$$w_i = \sqrt{K\left(\frac{d(\boldsymbol{x}_i, \boldsymbol{q})}{h}\right)} \qquad (5)$$

$K()$ = Kernel Function
$d()$ = Distance function
q = the query, a row vector of inputs to compared against the existing dataset.
h = fixed bandwidth

$$K = exp\left(-\frac{d(x_i, q)^2}{h}\right) \qquad (6)$$

The kernel function used is this implementation is called a Gaussian kernel. This function was selected because it has smoothing properties and is defined for all values of the distance function.

$$d = \sqrt{(x_i - q)^T (x_i - q)} \qquad (7)$$

The distance function used in this implementation is the unweighted Euclidean distance function because it is ideal for ordered, discrete input values which is what the inputs for the dataset created above consists of. In this implementation, the bandwidth was set as a fixed parameter that aids in smoothing the output. As h decreases, it results in a smaller weight for close distances, increasing the value of the kernel function. Decreasing h too much will cause the model to "overfit" meaning it is no longer is a good estimator of the underlying function. So finally in order to use these equations to make predictions the final equation is as follows:

$$\hat{y} = \boldsymbol{q}^T (\boldsymbol{Z}^T \boldsymbol{Z})^{-1} \boldsymbol{Z}^T \boldsymbol{v} \qquad (8)$$

$\hat{y}$ = the predicted output
$Z = W*X$
$v = W * y$

In order to help validate the predictions made by the LWR algorithm the following equations were used to compute the variance for each query point with respect to the locally linear model:

$$\sigma^2(q) = \frac{\sum_i r_i^2(q)}{n_{LWR}(q) - p_{LWR}(q)} \qquad (9)$$

Where:

$$r_i(q) = z_i(q)\beta(q) - v_i(q) \qquad (10)$$

$$n_{LWR}(q) = \sum_{i=1}^{n} w_i^2 \qquad (11)$$

$$p_{LWR}(q) = \sum_i w_i^2 z_i^T (Z^T Z)^{-1} z_i \qquad (12)$$

r_i = weighted residual
n_{LWR} = modified measure of the number of data points
p_{LWR} = the local number of free parameters in the local model

The (q) notation following some of the variables signifies that these values will most likely change for a given query point.

The dataset used to perform a locally weighted regression should contain 2 subsets. One subset is call the training data, and the other is called the testing data. The training data is a list of inputs that correspond to a list of known outputs. The testing data is also a list of inputs which also have corresponding outputs, but only the inputs are used in the actual regression. The known outputs in the test data set are used to evaluate the predictions of the regression. Because this method uses training data with known outputs, this method falls under the category of supervised learning. Also, by inspection of the equations above, the weighting matrix will change depending on the query so that means that the model is not trained prior to receiving a query point. This attribute causes LWR to fall under the category of lazy learning, because there is no training prior to receiving an input.

In order to validate that the implementation was functioning properly, a test case was run using the sine function. The input training data was created by using a list of values, θ_i, from 0 to 2π sampled at a regular interval and the corresponding output was calculated to be $\sin(\theta_i)+\varepsilon_i$, where ε_i is random noise from a Gaussian distribution. In this case, the underlying function is known to be $\sin(\theta)$ so it is expected that given a list of query points, the regression should be able to predict this function.

To test this hypothesis, a test set was created consisting of random values sampled from 0 to 2π. Each of these values was run through a regression model using the equations shown above for multiple bandwidths and the results are shown below in Figure 4.

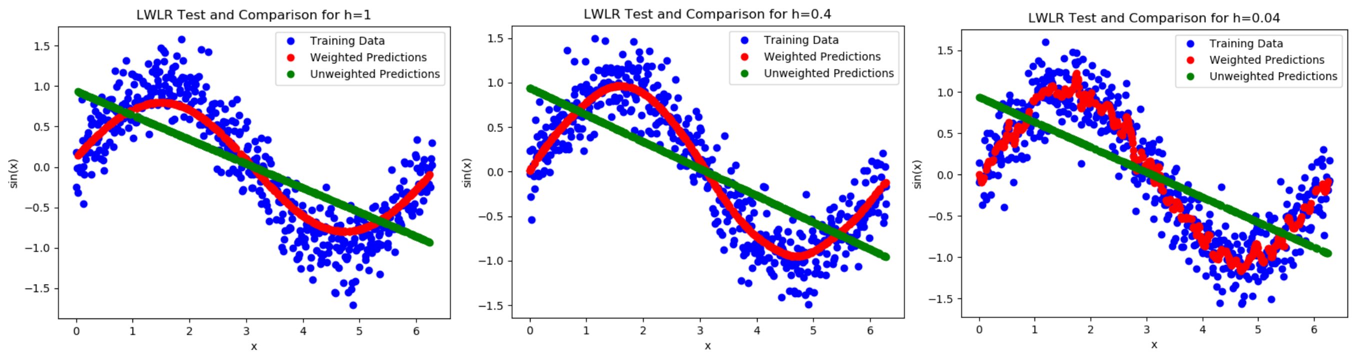

Figure 4. Testing results for the Locally Weighted Regression using multiple bandwidths

For this test, the underlying function is known to be $\sin(\theta)$ with an amplitude of 1, so ideally the regression model should reflect this function. Using a bandwidth of 1, yields a graph that displays a proper looking sine wave, but the amplitude is less than what is expected. By decreasing the bandwidth, this increases the weight assigned to points near the query point producing an estimate that more accurately represents the underlying function. However decreasing h too far will results in over-fitting as shown in the third plot. While the estimate still has a sine-like shape, the estimate of the underlying function is being distorted by the noise in the training data.

Another method of evaluation can be to look at the variance for each query, which is shown in Figure 5 for each test. The variance for a bandwidth of 1 and 0.4 are both consistent across the entire test set, but the 0.4 test yields a smaller average variance meaning the query point varies less compared to the local linear model. The bandwidth of 0.04 does results in some sections that have an extremely low bandwidth, but it also creates sections where the variance is much larger, resulting in a higher average variance than the 0.4 test and thus a worse overall estimate of the underlying function.

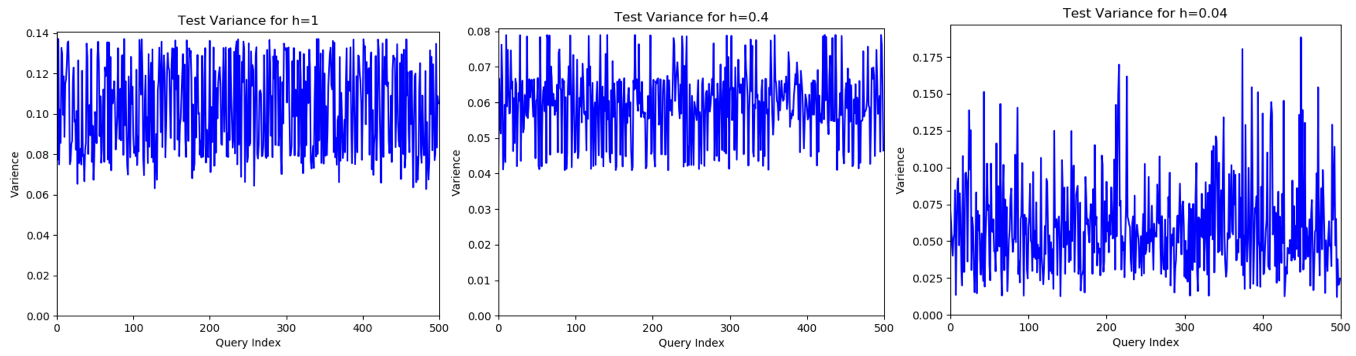

Figure 5. Variance Results for the sine wave regression test

Questions 3: Algorithm Applied

The dataset formulated in part 1 was then applied to the algorithm developed in part 2. The full dataset created in part 1 consisted of ~12,000 data points. For each test below this data was split into two subsets; the test dataset consisted of 1,000 consecutive data points leaving just under 11,000 in the training dataset. The same global bandwidth value was used across all tests.

For both tests, the first group of plots in Figure 6 and Figure 9 compare are similar to those presented in part 1. These plots were used to visually inspect the direct performance of the LWR algorithm at predicting the change in each state variable.

The second group of plots in Figure 7 and Figure 10 show the predictions of the LWR compared against the known values from the groundtruth data and the odometry trajectory. The left plot is a calculation of using the predicted state changes to simulate the robot being controlled according to the LWR output. All three trajectories start at the same location and then propagate forward. The middle and right plots are the absolute error for each prediction:

$$e_q = \left|\widehat{y_q} - y_q\right| \qquad (13)$$

Also recall that the expected output and predictions have both been scaled to a uniform time step of 1 sec.

The third grouping of plots in Figure 8 and Figure 11 show the variance calculation for each test data point using the equations discussed in part two.

After analyzing the figures, it is clear that this implementation of LWR would not be able to function as a reliable controller for this application. Fundamentally, the LWR is creating a bunch of local linear fits to the training data set in order to estimate the desired output value, in this application a change in state. In this case, even though some of the initial plots seem to imply a locally linear relationship, the plots comparing *v* and *ω* to each state change imply there is a wide noise distribution across the input space. Averaging across a distribution will yield a nice curve, but when trying to predict an exact path, the average could be quite far from the actual state. This effect can be observed using the absolute error and variance plots for each prediction. In general, when there is an increase in the variance, the error in the prediction also increases, meaning that this query point is not close to the local linear model. Also, looking at the training and testing data plots in Figure 6 and Figure 9, it can be seen that a similar set of inputs in the training data could potentially yield a drastically different change in state. So when the algorithm attempts to fit the locally linear model to this region, it will result in a model that is in-between the two values, resulting in a prediction that is between the two training points. An improvement to this implementation could be to incorporate local weighting using the PRESS statistic and Robust Regression to identify which of these points should be classified as outliers.[2] A particularly useful application of this would be on the data describing the change in *θ* as some of the time-scaled *dθ* values were +10 rad (off of the current grid scaling) which resulted in some very large variances at those locations. Also reviewing the plots in Figure 3, there seem to be a significant amount of points in areas where they should not be possible (i.e. if the robot is facing left, *dx* should be less than or equal to 0). However, it is unclear if these data points should be considered outliers since some are quite close to the main distribution; they may represent unique scenarios. When a linear model is fit to this area it will be slightly shift due to this outlying but valid data point. Some of this error could have also been introduced by the assumptions made in the dataset formulation.

For the UTIAS study, the robot was free to move anywhere about the course based on the landmarks and other robots it was able to detect with the onboard camera. However, a scenario that might yield better predictions from an LWR would be for a robot that was programmed to repeatedly traverse a set path. One application of this would be a patrol path for a security/surveillance application or a robot that is moving around in a sorting facility between various pick-up and delivery locations. This type of behavior would most likely yield a data set with a tighter spread across the dataset yielding an "average" path that is closer to the underlying path.

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

Although the overall predictions did not cumulatively stack up to predict the trajectory well, there were some sections of the dataset that did perform quite well. Comparing the variance and error plots for each test, when the variance was near zero, the prediction error was ~1cm for a time step of 1 sec. From the odometry data, the actual time step between control commands was ~01. – 0.2 sec. meaning the actually difference in the state change and the prediction for *x* and *y* was ~1-2mm. However, this mix of good and poor performance could also point to data set issues. Some of the poor behavior could be due to outliers as discussed previously, but there also could be issues of data deprivation for that region of the input space. Looking at the plots for the change in *x* in Figure 6, the predictions made when $v = 0$, had very little information to base the prediction on, thus the resulting output for those points was a negative *dx*, when the actual change was near zero.

Another feature of the LWR is that it was good at following the overall pattern of the groundtruth trajectory, even though the actual state values may not be very similar. By inspecting the different trajectories on the plots in Figure 7 and Figure 10, the trajectory of the cumulative predictions captures the features of the ground truth trajectory very well (i.e. turns, figure-8's, straight lines, etc.). This also points to the fact the algorithm is preforming well given the training data it has to make predictions on. It is identifying the underlying trajectory, but, due to the training data, the local linear fits are offset from the actual underlying function creating error in the prediction. Something that may substantially improve the results is just more quality data points to fill in gaps in the input space and to help reveal what behavior should be considered as an outlier.

Bibliography

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

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.

TEST 1:

The testing data used was from index 550 to 1550 of the full dataset.

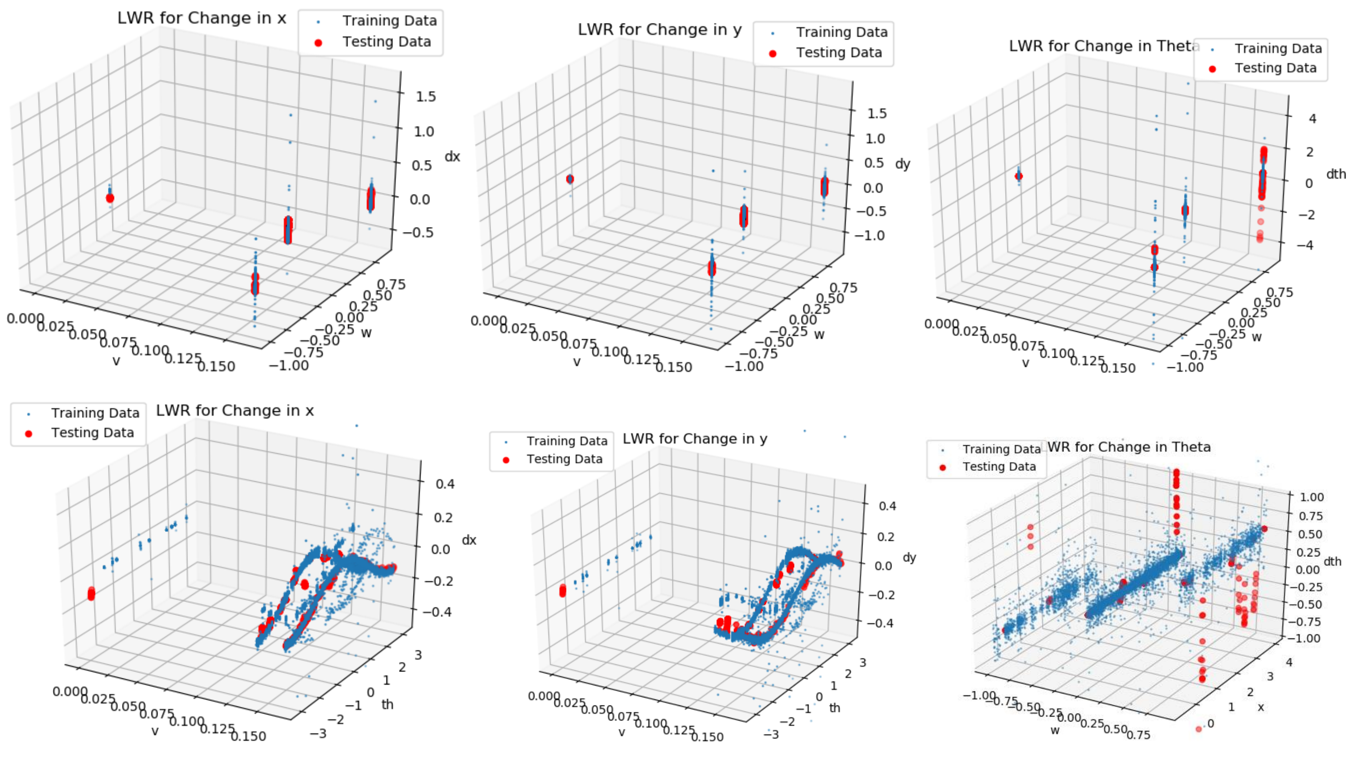

Figure 6. Training data plotted alongside testing data inputs and predicted outputs

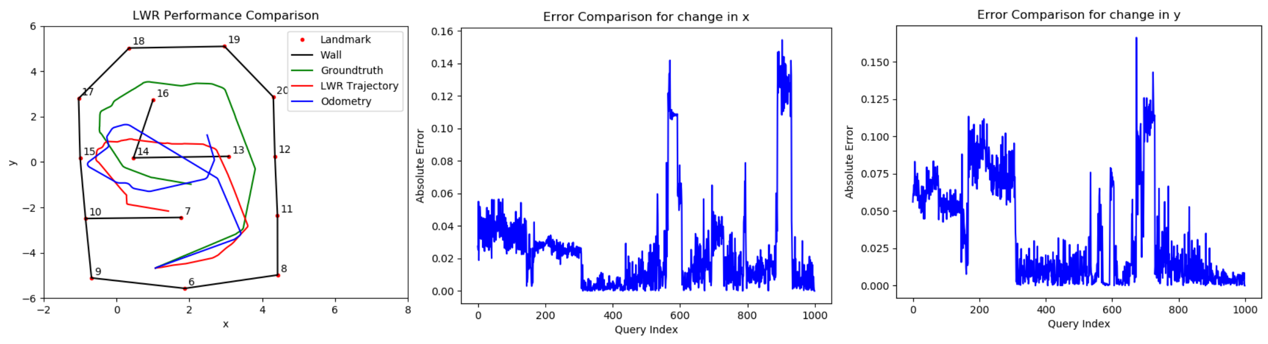

Figure 7. LWR trajectory comparison and Error in dx and dy predictions

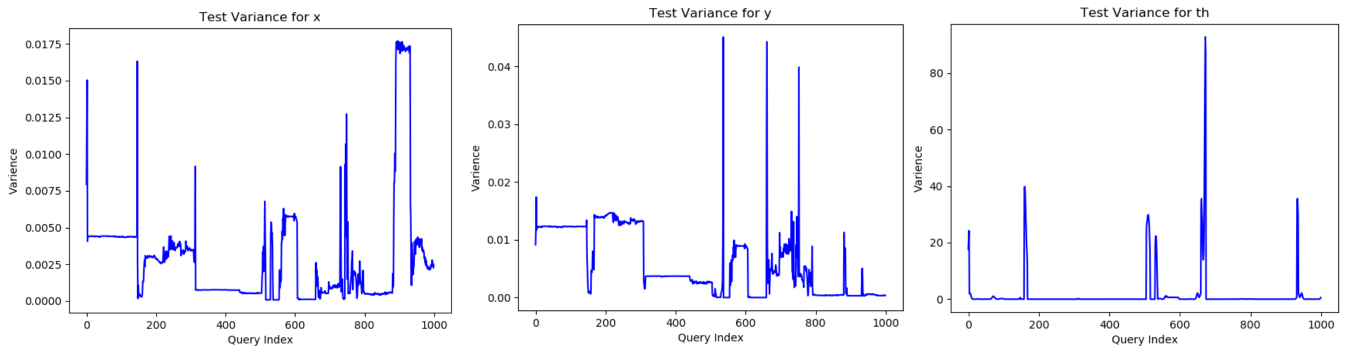

Figure 8. Variance calculation for each test data point

TEST 2:

The testing data used was from index 3550 to 4550 of the full dataset.

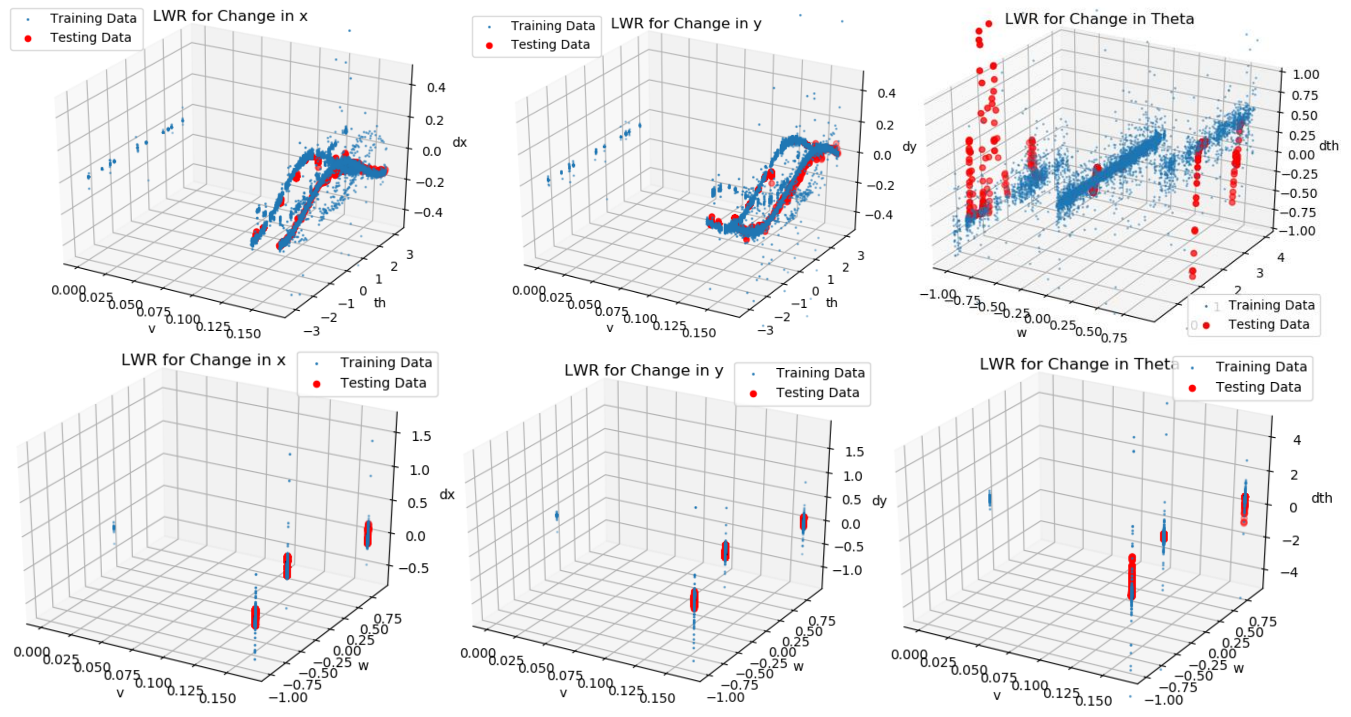

Figure 9. Training data plotted alongside testing data inputs and predicted outputs

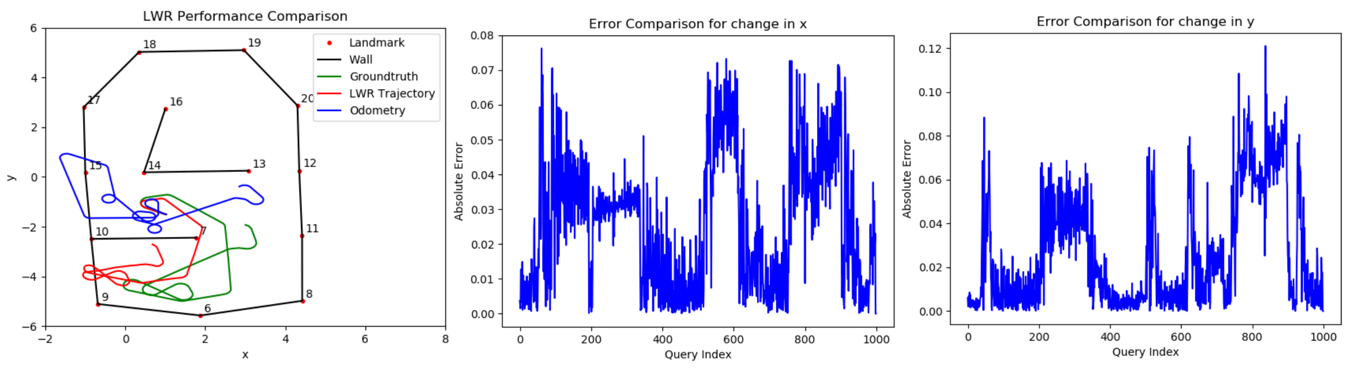

Figure 10. LWR trajectory comparison and Error in dx and dy predictions

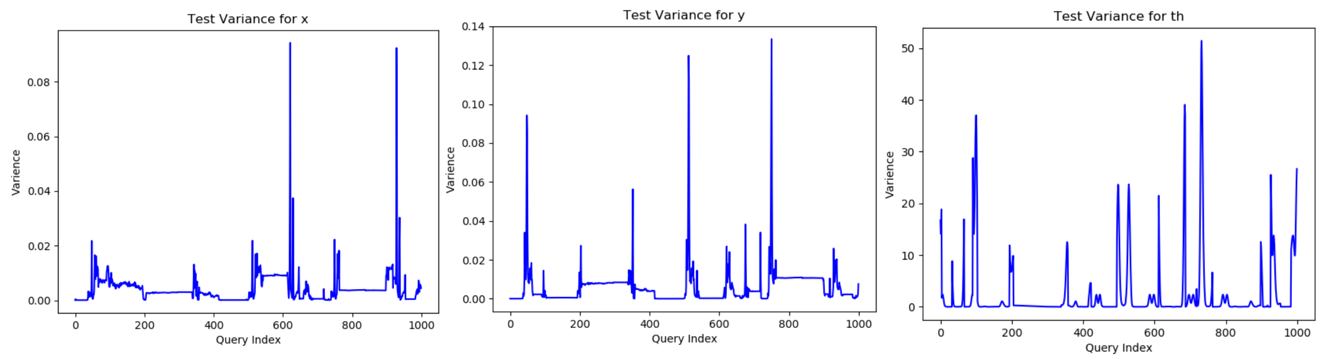

Figure 11. Variance calculation for each test data point