

Implementation and Assessment of Learning Algorithm: Gaussian Process Regression

Approval due: Nov. 5th • Code A due: Nov. 10th • Code B due: Nov. 15th
Writeup due: Nov. 15th

Mackenzie Gray

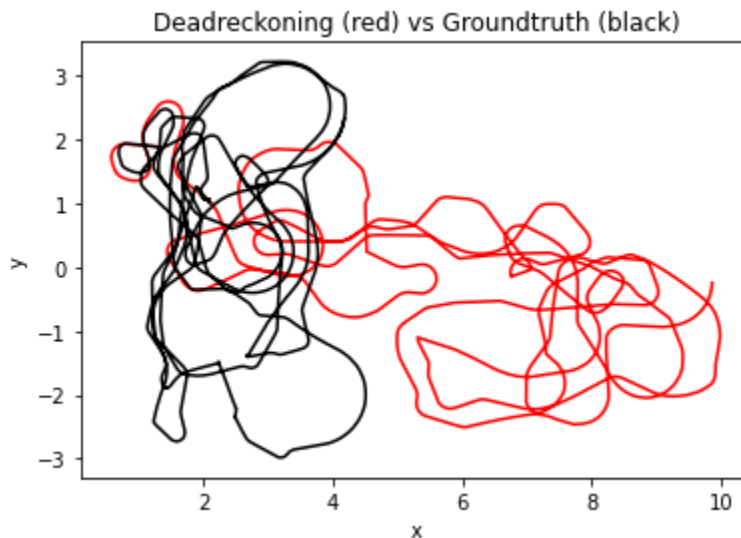
PART A:

1. Build a training dataset

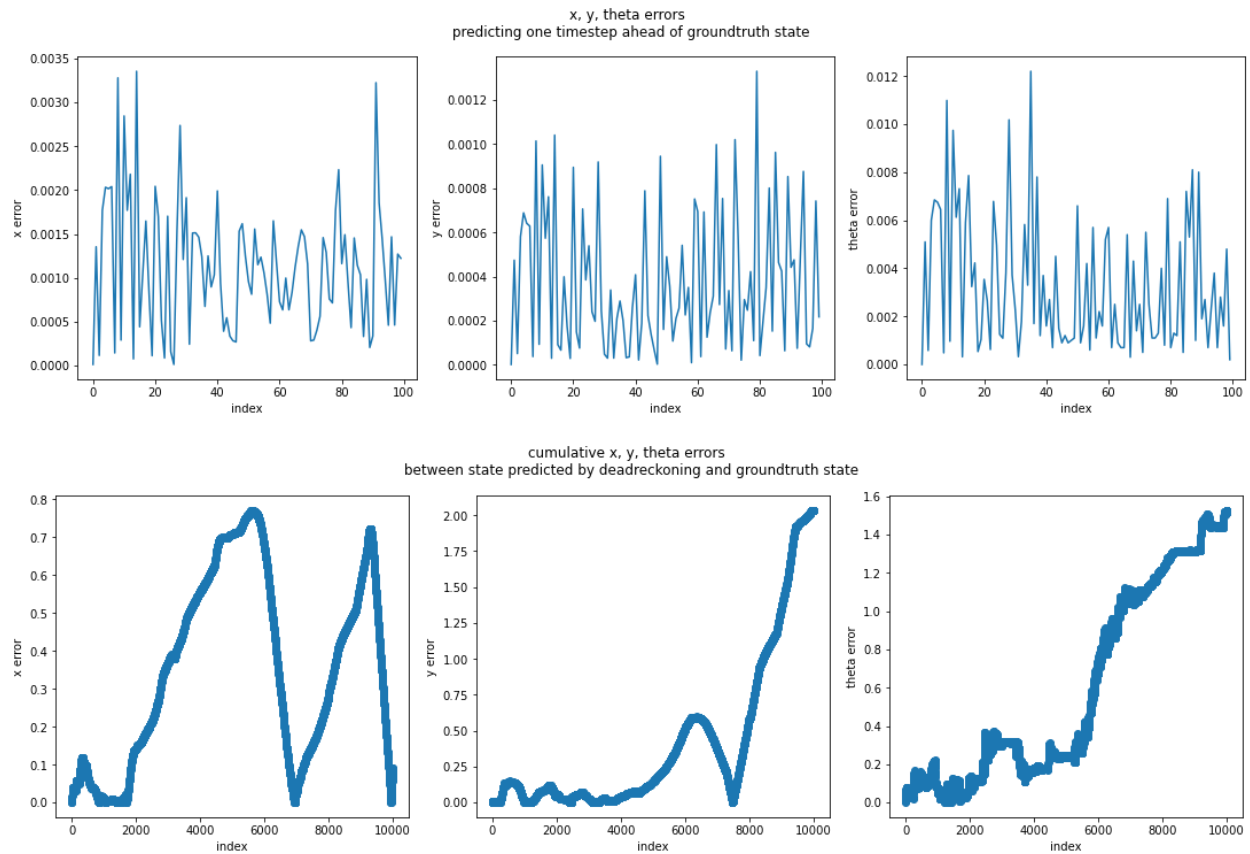
In Assignment 0, the following kinematic equations were used to model the motion of the mobile robot given translational and rotational velocity commands:

$$\begin{aligned}x_{t+1} &= x_t + (-v_{\text{command}}/w_{\text{command}})*\sin(\theta_t) + (v_{\text{command}}/w_{\text{command}})*\sin(\theta_t + (w_{\text{command}}*dt)) \\y_{t+1} &= y_t + ((v_{\text{command}}/w_{\text{command}})*\cos(\theta_t) - (v_{\text{command}}/w_{\text{command}})*\cos(\theta_t + (w_{\text{command}}*dt))) \\ \theta_{t+1} &= \theta_t + (w_{\text{command}}*dt)\end{aligned}$$

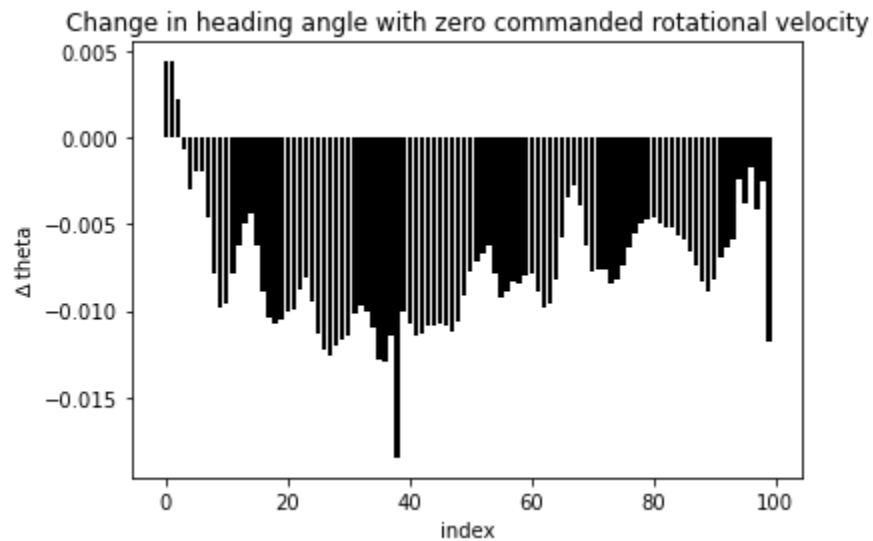
The equations assume that the robot travels at the commanded velocities for the duration of the timestep and thus determine the robot's x,y position and heading angle, theta, at the next timestep given its current state. This model, however, does not account for a number of inaccuracies in the real motion of the robot which can be shown visually by comparing a plot of the groundtruth path of the robot to the dead reckoning path based on this idealized set of equations.



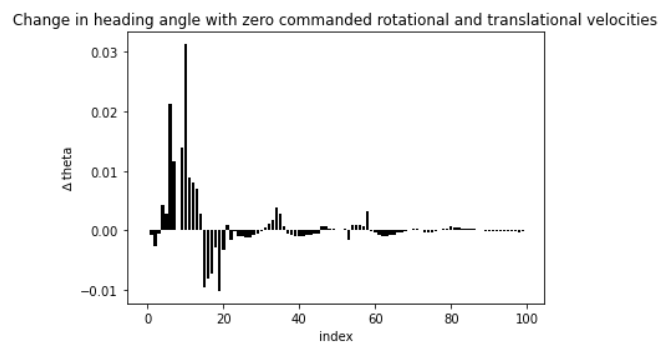
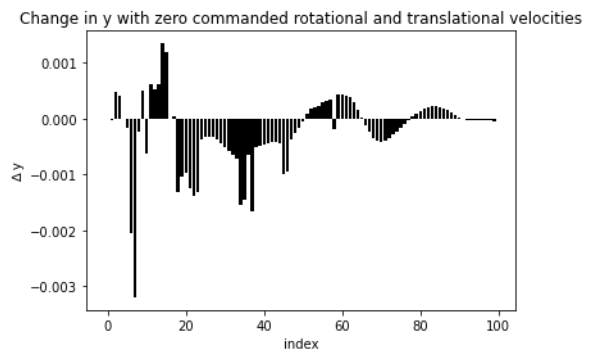
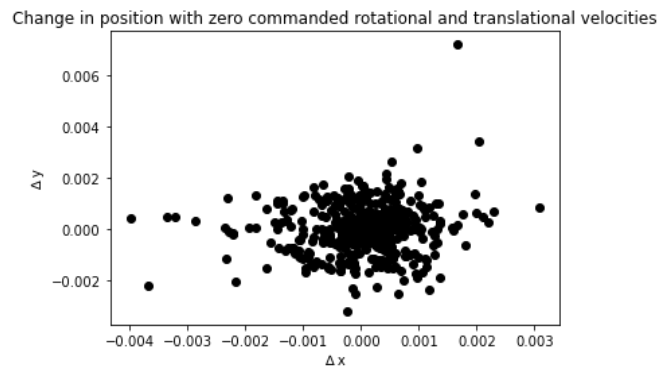
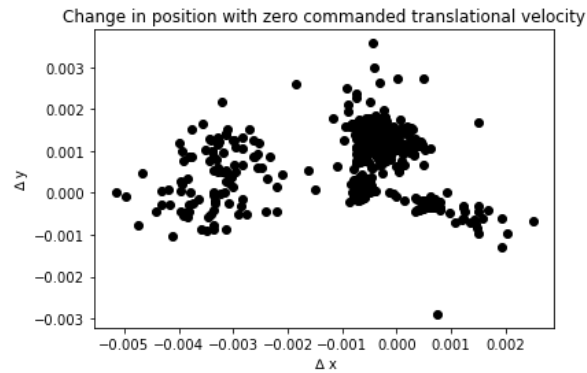
These inaccuracies may arise from wheel slip, axel misalignment, and discrepancies in power delivered to the right vs. left sided wheels by separate motors. The model also fails to account for acceleration limits; it assumes that the robot attains the velocities it is commanded, however, if the velocity commands differ greatly from the current velocity of the robot and the timestep is small, the robot is likely not able to reach the commanded velocities during the timestep and thus the true change in state will lag that predicted by the model. Overtime, these errors will accumulate and each subsequent state prediction will drift further from the groundtruth state of the robot.



Based on the equations above, when the commanded rotational velocity is zero, the robot is expected to travel in a straight line, therefore, the change in heading angle is predicted to be zero. In the plot below, the groundtruth data shows this is not the case.



Additionally, when the commanded translational velocity is zero, the robot is expected to rotate in place, therefore the change in position (x and y) is predicted to be zero. Again, the plots below show that the true position of the robot does change which makes sense given that the robot will need time to slow down and come to rest if it is moving when the zero velocity commands are first received.



In order to improve the motion model and account for prediction errors caused by the discrepancy between the robot's actual and commanded velocities, I used these differences as inputs from which to predict the change in state of the robot to be learned through Gaussian Process Regression.

A Gaussian Process is a non-parametric machine learning technique that expresses a probability distribution over functions defined by a mean and kernel function. Once trained, the distribution of possible functions is constrained such that the mean passes through all training data points. The standard deviation, which governs the spread of the distribution, is higher away from training data and reflects the model's lack of knowledge of these areas, thus increased uncertainty in predictions.

The kernel function models the covariance between each pair of input points and results in a matrix that is positive definite and is therefore symmetric and invertible. While numerous kernel functions fit this criteria, the Gaussian or Radial Basis Function kernel is popular and used in a wide range of practical applications. It can be computed as follows:

$$k(x_1, x_2) = \sigma_f^2 * \exp(-\|x_1 - x_2\|^2 / 2\ell^2)$$

Where σ_f^2 represents the variance and ℓ the lengthscale

$$K_{xx} = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \dots & k(x_2, x_n) \\ \dots & \dots & \dots & \dots \\ k(x_n, x_1) & k(x_n, x_2) & \dots & k(x_n, x_n) \end{bmatrix}$$

The aim of the algorithm is then to predict the posterior distribution of test inputs conditioned on observed training points. The mean of the predicted outputs is also computed as a linear combination of observed outputs (target values) that are assigned weights based on the kernel distance between training and test inputs.

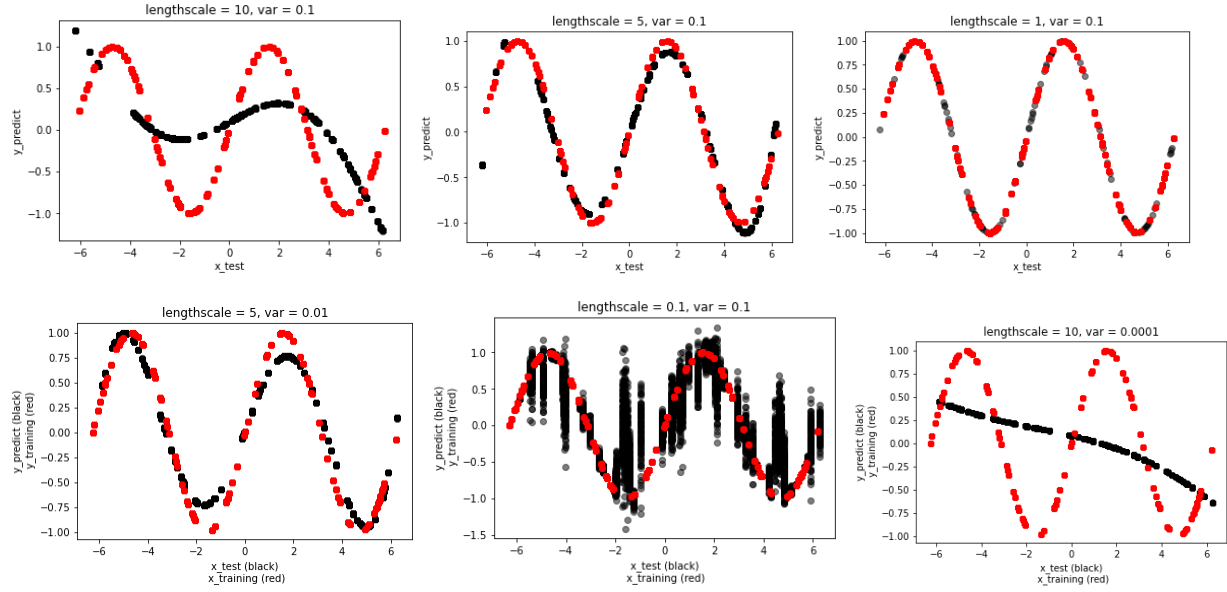
Given that GPRs are non-parametric, they do not learn a large set of parameters and can achieve strong generalizability to unseen test data from a smaller set of training data than other techniques such as neural networks. Additionally, GPRs estimate both expected values and uncertainties of these predictions which may be taken into account and leveraged in decision making steps crucial to the success and efficiency of model-based reinforcement learning with applications in policy optimization and design of a controller.

Given the set of training inputs, x , training outputs, y , and test inputs x_* , the mean and covariance of the predicted outputs can be computed according to the following equations:

$$\begin{aligned} \bar{f}(x_*) &= k(x_*, x) [k(x, x) + \sigma_n^2 I]^{-1} y \\ \text{Cov}(\bar{f}(x_*)) &= k(x_*, x_*) - k(x_*, x) [k(x, x) + \sigma_n^2 I]^{-1} k(x, x_*) \end{aligned}$$

In the code, Cholesky decomposition along with scipy's `linalg.solve_triangular` function were used to avoid the potential numerical instability that may arise from matrix inversion.

Upon implementation of the algorithm, its proper functioning can be confirmed by testing on a known function. In this case, a sine function was used.



The above plots show that the fit of the posterior distribution increases (improves) as the lengthscale parameter involved in calculating the kernel function decreases. However, if decreased too far, the data fit performs poorly as shown in the bottom left plot.

To evaluate the performance of the learning algorithm, the sum of squared errors and log marginal likelihood were calculated. A smaller sum of squared errors and larger marginal likelihood reflect better predictions of the outputs from the posterior distribution. The marginal log-likelihood can be conditioned on the lengthscale, ℓ , prediction variance, σ_f^2 , and noise, σ_n^2 . Then using an optimization scheme such as gradient descent, the negative marginal log-likelihood can be minimized with respect to these hyperparameters to tune their values based on the observed data. The marginal log-likelihood is defined by the following equation:

$$\log p(y | x, \ell, \sigma_f^2, \sigma_n^2) = -\frac{1}{2} y^T K_y^{-1} y - \frac{1}{2} \log |K_y| - (n/2) \log(2\pi)$$

Where $K_y = K + \sigma_n^2 I$

The first term can be interpreted as the data fit. The second as the complexity penalty. And the third as a normalization constant. The data fit decreases with the lengthscale as the model becomes less flexible while the complexity penalty increases given that the model becomes more complex as the lengthscale shrinks.

To calculate the sum of squared errors, outputs were sampled from the posterior distribution using numpy's random.multivariate_normal function along with the mean and covariance as determined from the gaussian process algorithm for the given training data and test inputs. These predicted values were compared to the true output values for this set of test input data and the sum was calculated as follows:

$$\sum_{i=0}^N (y_{\text{true}, i} - y_{\text{predict}, i})^2 \text{ where } N \text{ is the total number of test points}$$

PART B

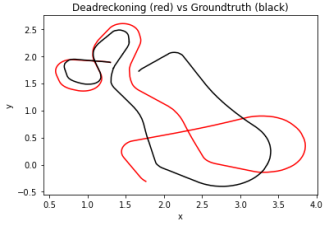

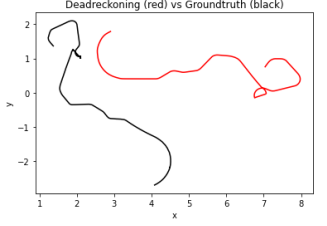
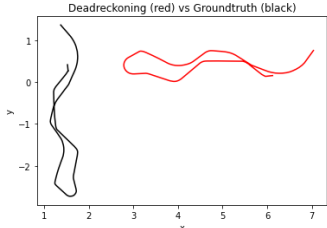
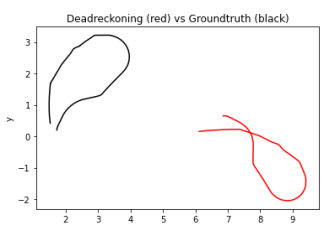
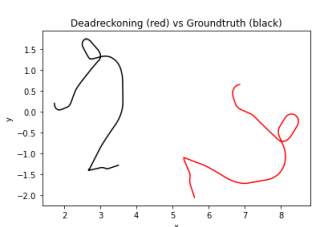
3. Apply learning algorithm to dataset and evaluate its performance using at least two measures

A number of different combinations of inputs were tested to examine which provided the best performance for predicting the change in state variables (x, y, theta). The below table shows the performance metrics for predicting the change in x given the difference between the robot's current and commanded translational and rotational velocities and duration of the timestep as inputs to the algorithm. The bottom row shows results using the optimal fit lengthscale parameter as determined by the sklearn GPR library and associated functions. The fit function provided by this library minimizes the negative log likelihood to determine optimal parameter values.

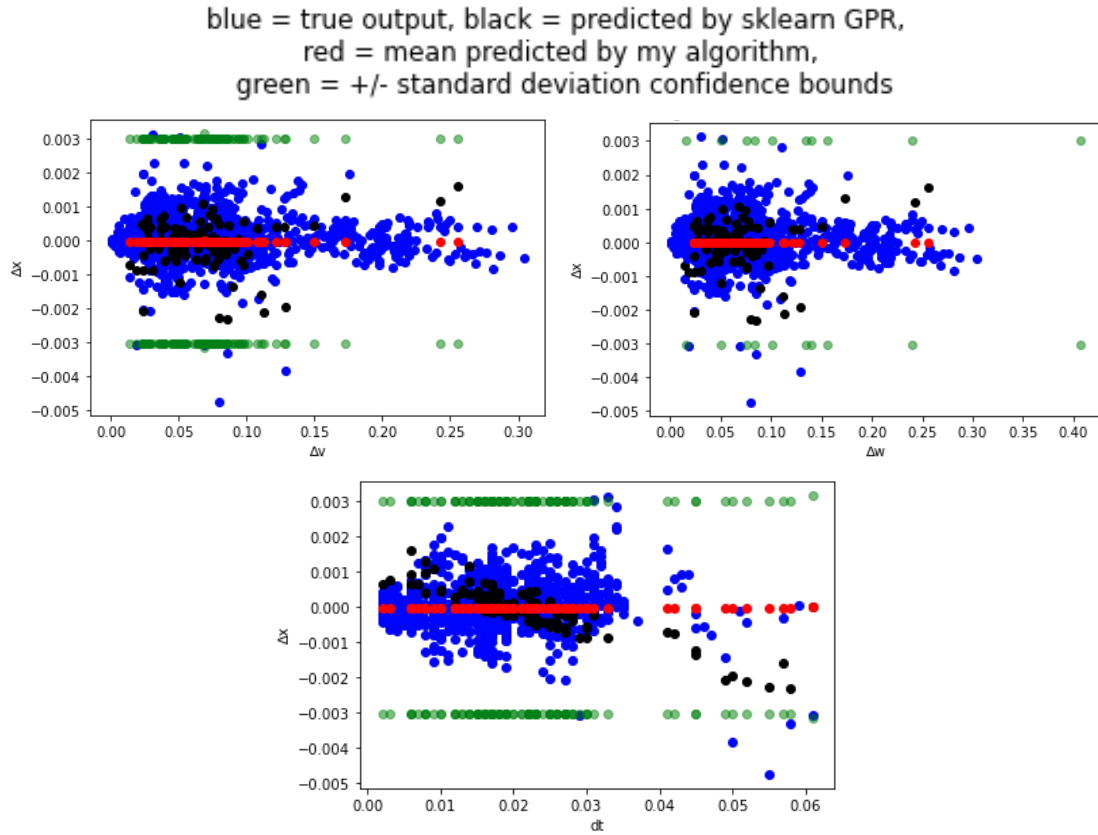
As expected, the log marginal likelihood is maximized for these optimal parameters. Additionally, the average squared error (given by the sum of the squared errors divided by the number of test points) decreases as the number of training points increases. This is to be expected given that all points are used to compute the posterior distribution from which predictions are made.

Drawbacks to using gaussian process regression on this learning set stem from its computational complexity; the time complexity scales as $O(n^3)$ and the storage as $O(n^2)$ where n is the number of training points. To circumvent the need to invert large matrices, the cholesky decomposition was utilized, however, this limited the ability to modulate the noise parameter over a wide range of values because of the need for the covariance matrix to be positive semi-definite. Additionally, each output variable was predicted independently, so the model misses the interdependence of changes in x, y, and theta.

# Training Points	# Test Points	Length Scale	Variance	Average Squared Error	Log Marginal Likelihood
100	10	1	0.1	0.00213	20.0544
100	10	10	0.1	0.00133	20.9105
100	10	100	0.1	0.00089	20.9276
100	10	1000	0.1	0.00030	20.9278
100	10	10000	0.1	0.00013	20.9278
1000	100	100	0.1	0.00060	228.8964
10000	1000	100	0.1	0.00020	2318.5657
1000	100	21.6	0.00001	0.0000116	2323.1935

Data Index Interval	Average Squared Error (5000 Test Points)	Log Marginal Likelihood	
0:10000	12.7028e-5	2317.5394	
10000:20000	22.4517e-5	2317.9386	
20000:30000	6.5214e-5	2317.1910	
30000:40000	7.1355e-5	2317.9405	
40000:50000	6.6823e-5	2317.9046	
50000:60000	11.2025e-5	2317.5709	

The table above displays the performance metrics of the algorithm given different intervals of 10000 training points along with plots of the deadreckoning vs ground truth paths. The algorithm predicts the change in x, so it makes sense that the smallest errors correspond to the intervals over which the robot is moving primarily in the vertical direction. It is interesting to note that the dead reckoning paths do closely resemble the appropriate shape of the paths but are rotated by 90° clockwise. In the case of the interval from 30000:40000, the dead reckoning path predicts that the robot is traveling horizontally when in fact it is travelling vertically. This may cause physical damage to the robot if used in a model-based optimal policy search as it may result in an unexpected collision with an obstacle. Therefore, using the model learned by gaussian process regression would be a better option.



The above show the true change in x coordinate plotted against each of the input variables in blue, the mean output values as predicted by the sklearn algorithm in black, the mean output values as predicted by my implementation of the algorithm in red, and the confidence interval given by the standard deviation of the posterior determined by the square root of the diagonal entries of the covariance matrix defining the posterior gaussian distribution.

The confidence interval does succeed in capturing the spread of the data, however, the data is very clustered which makes it challenging to interpret in relation to the physical dynamics of the robot being modeled. The data set would be improved with greater diversity in the inputs and greater uniqueness of outputs. Additionally, the input data involves measurements taken from odometry and ground truth data sets which are logged at differing frequencies using separate hardware, so inaccuracies can also be attributed to how the timesteps were aligned in constructing the datasets used for training and testing the

algorithm. To improve learning performance, more intentional driving could be employed. This would include commanding fast velocities while the robot is at rest such that the robot reaches its acceleration limit. Additionally, constant commanded velocities should be issued for longer durations to reflect steady state driving dynamics. Lastly, given that the data set calculates the instantaneous velocity of the robot from ground truth data from a 10-camera Vicon motion capture system with accuracy on the order of $1 \times 10^{-3} \text{m}$, a better option would be to use a speed sensor or imu to capture the velocity directly. Taking averages over longer timesteps may also lessen the effects of errors incurred due to limits on the accuracy of the data.

Additional Resources:

<https://towardsdatascience.com/gaussian-process-regression-from-first-principles-833f4aa5f842>

<https://github.com/ebonilla/gaussianprocesses/blob/master/notebooks/gp-inference.ipynb>

<http://www.gaussianprocess.org/gpml/chapters/RW2.pdf>

<http://proceedings.mlr.press/v5/titsias09a/titsias09a.pdf>