Ryan King-Shepard
ME469
Assignment 2

Support Vector Regression Write-up

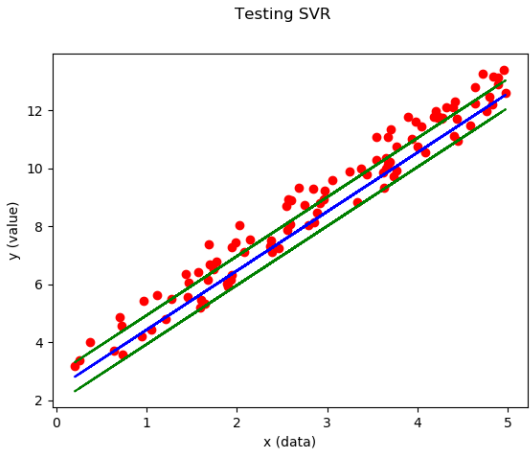
For assignment 2, my learning goals are to use support vector machines for regression in order to re-create the measurement model of the robot. I compared the performances of various kernels to see which one is able to fit the measurement model best, which produces the minimal error when compared to the ground truth data, and explored if the grid contained high-error areas invariant to the kernel model.

For the training data of this problem, I'm crafting a dataset of 5 dimensional features with 2 single-dimensional label vectors. The label vectors are the range and relative bearing of the robot from known landmark. The feature vectors will contain the current robot's pose, it position and current heading, and the landmark's position. The goal of the support vector regression will be to learn functional mappings from the robot's pose and landmark's position to the range and bearing measurement.

Support vector regression is a supervised learning algorithm that attempts to learn a model that can accurately predict a y_q value when given a query feature x_q . The shape of the model is determined by it's kernel, an inner product of the feature set. Kernel functions can be mapped into a higher dimensionality that allows support vector regression to even learn non-linear model. For example, a simple linear kernel will attempt to fit a hyperplane to the dataset. If the dataset present non-linear characteristics, kernels such polynomial or radial basis can find a linear model in higher dimensions which will map to a non-linear model in the original feature set dimensions. Since the relationship between the robot's pose and landmark's position to the range and bearing are not linear, the support vector regression algorithm was chosen in order to test various non-linear kernel for performance comparison.

The support vector regression algorithm works by solving for Lagrangian multipliers for the minimum cost. The dot product of the difference of the multiplier solution and the feature set form the weight vector of the model. These multipliers categories each feature-label pair into one of three groups: either the feature-label pair is outside the acceptable error tolerance ε, the feature-label pair is within the error tolerance, or the feature-label pair is perfectly predicted by the model. The pairs that fall into the second and third group are known as support vectors, excellent representatives of the data used to determine the intercept of the model. These two values form a model of $y_q = \langle w * x_q \rangle + b$ where w is the weight vector and b the intercept.

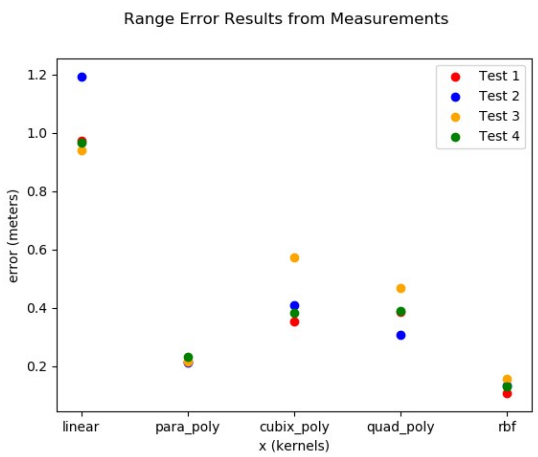
To give an example of the implementation functioning for linear regression, below is a plot of support vector regression mapping to the line 2x+3 with an ε value of .5

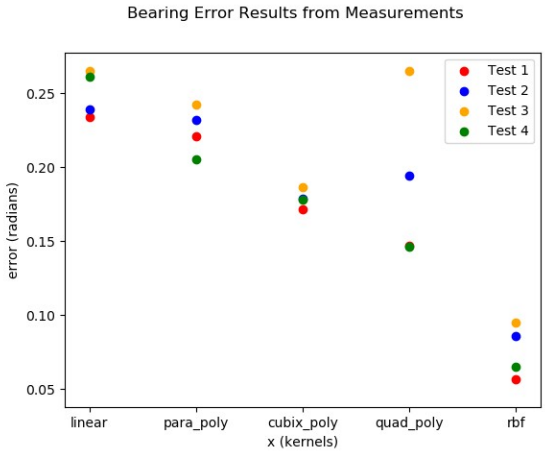


[Explain Support Vector regression]

For the application of the support vector regression, 5 candidate kernels were chosen to compare performance: a linear model, 3 polynomial models representing the orders of two, three, and four, and a radial basis kernel. **A note for the grader: due to debugging errors and difficulty with using a quadratic programming solver, only the results of the linear model were from my implementation. The other kernel results are from the python library sklearn.**

The learning process for the measurement model used cross-validation. The dataset was split into four parts; three of the parts became training data, the final part used as testing data.





Each kernel was given the same slack penalty of one and used an ε of 0.0705 for learning the range and an ε of 0.017453 for learning the bearing. The slack penalty was chosen based on several reading examples and implementation trial-and-error, but the εs have design motivation behind them. The range ε is half the diagonal distance for a cell in a discretized grid of 0.1. This allows the insensitive tube of range to represent a radius around a center point where the robot is considered to be in the same location for all points in the circle formed. The bearing ε represent a one degree tolerance, given the bearing measurement a two-degree window centered around the expected value. The above graphs shows the error resulting from each validation test for the 4 different sets of data.

The expected functions we hope the kernels will approximation should resemble the calculations for range and bearing. As a reminder:

$$range = \sqrt{((robot_x - landmark_x) + (robot_y + landmark_y))}$$

$$bearing = \arctan((landmark_y - robot_y)/(landmark_x - robot_x)) - robot_\theta$$

For the linear performance, the kernel calculation is just the dot product of the feature set and it's transpose.

$$linear\ model: \langle X * X.T \rangle$$

Given the shape of the linear model, it will have a difficult time approximating to functions resembling the original range and bearing calculation. In terms of both learning range and learning bearing, it is the worst regression model. It is mostly used as bench-mark and test case for implementation functionality.

The polynomial kernel is the same dot product of the linear model raised to a particular power.

$$polynominal\,model: \left(\langle X * X.T \rangle + A\right)^b$$

For the above results, instead of exploring values of A, three polynomial models of order two, three, and four were trained and tested. To understand how the shape of the kernel would effect result, the value of `A` was set to zero for all three options. The `para_poly` model represents the quadratic polynomial and does quite well for estimating the range. This is expected given that geometrically, a quadratic polynomial and square root function have a similar shape. This assertion would not hold for feature sets of low dimension; representing a square root function with a quadratic model in one-dimension would result in terrible performance. But as the number of dimensions increase, the parabolic nature of the two functions serves as a utility. Unfortunately, the utility does not apply when a

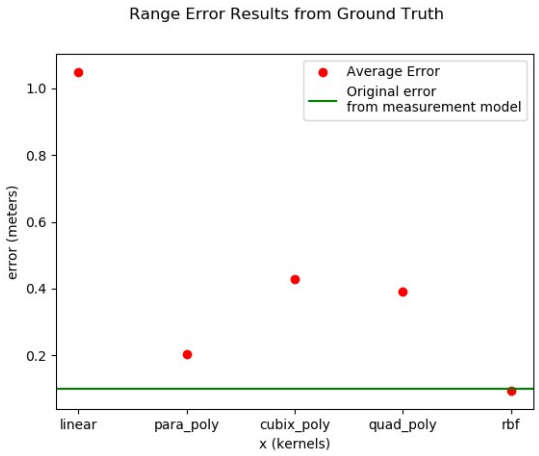
The `cubix_poly` and `quad_poly`, respectively represent the third and fourth order polynomial kernel. Despite these kernels mapping to higher dimensions, their performance compared to the quadratic polynomial is not that impressive. Their results also contains an odd spread in error. This is likely caused by the models over-fitting to the training data, a risk for higher order polynomial kernels. The kernel is a mapping from the low-dimensional space of our feature set to a higher dimensional space. The higher dimensional space introduces more features allowing the regression to fit to high dimensional model that is transformed into the low-dimensional space. However, if too many new features are introduced in the higher dimensional space, the regression may fit a model that only applies to the data presented resulting in poor test performance.

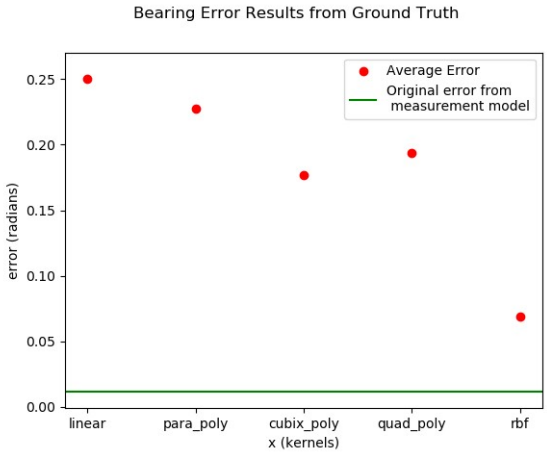
The radial basis kernel takes on the following form:

$$e^{-(\|X_i - X_j\|^2)/(2\sigma^2)}$$

where $\|X_i - X_j\|$ represents the space of one sample from the training set to all other samples. For our examples, σ is 1.

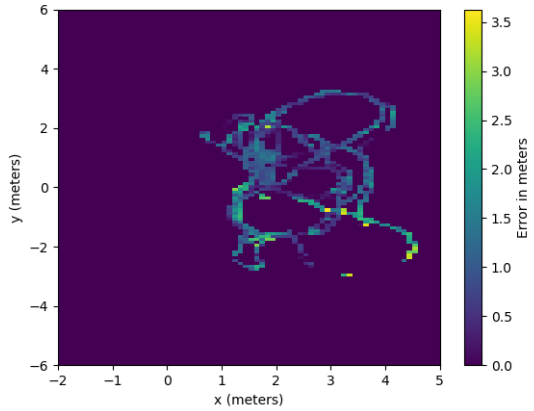
The kernels were also compared against the ground truth of range and bearing to see if one of the resulting models could produce less errors than the general measurement sensors.



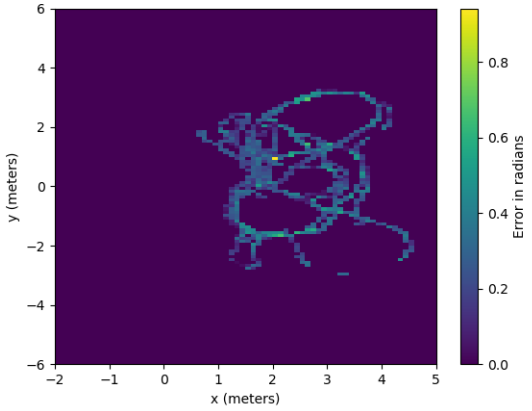


For the exploration of high-error areas invariant to the kernel model, I found that no such area existed. Below are a series of error-map focused on range and bearing error for each particular kernel.

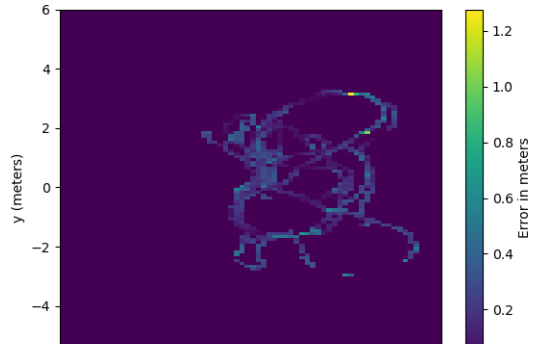
Error Grid for Range: linear kernel



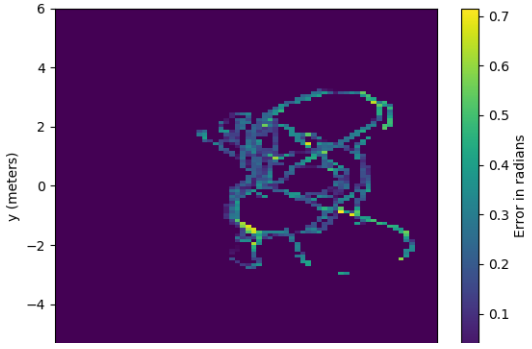
Error Grid for Bearing: linear kernel



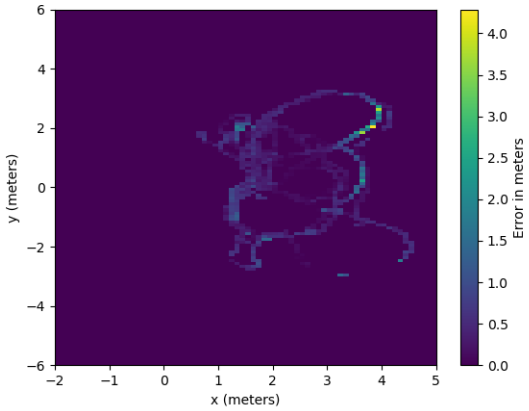
Error Grid for Range: parabolic (poly ^2) kernel



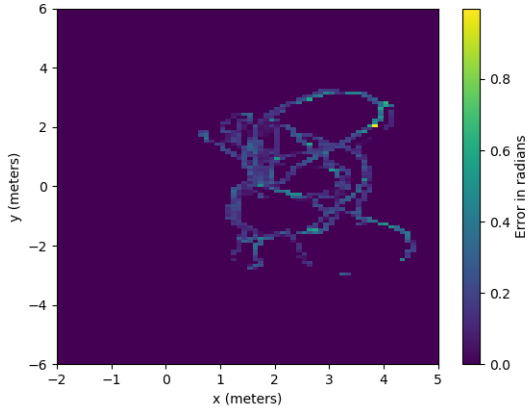
Error Grid for Bearing: parabolic (poly ^2) kernel



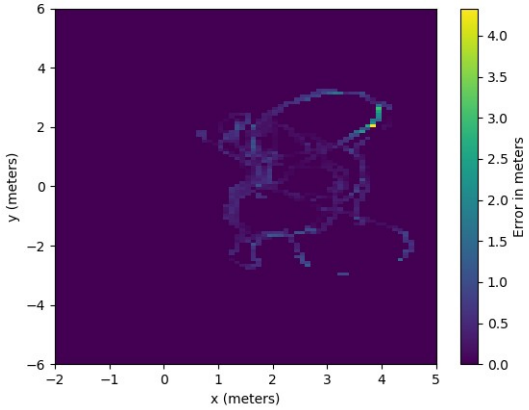
Error Grid for Range: cubix (^3) kernel



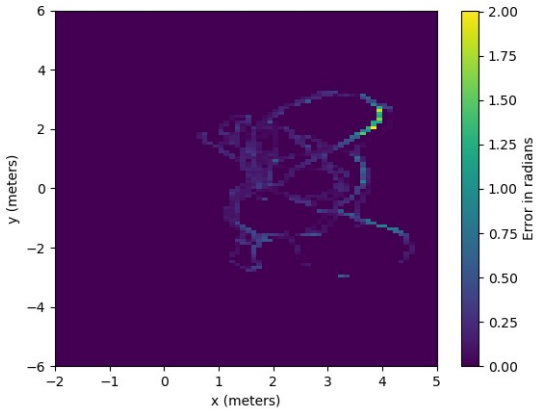
Error Grid for Bearing: cubix (^3) kernel

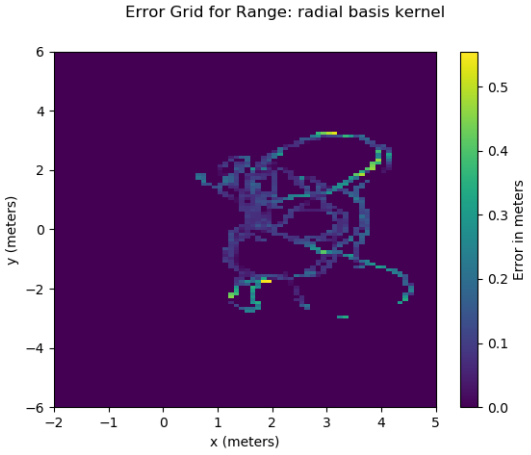


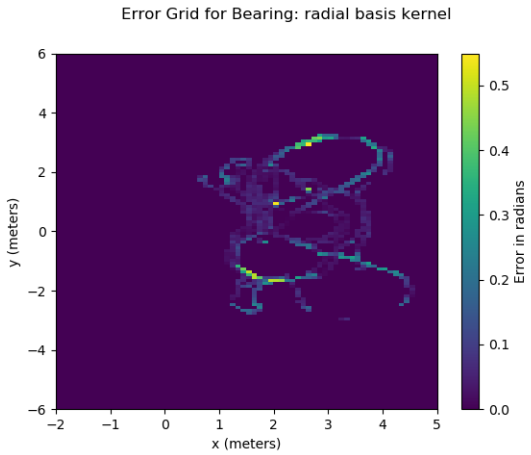
Error Grid for Range: quad (^4) kernel



Error Grid for Bearing: quad (^4) kernel







This question was meant to explore the idea that the topology of the grid would have certain locations where accurate measurement values would be difficult to obtain. Metaphorically, if the robot had consistent " bad reception" in certain regions, these regions would propagate into our kernel models, regardless of their performance, as region of relatively high error. However, each kernel model presents different regions of high error, which implies the issue based on the kernel's learning, not on the grid's topology.