CS8803-01 Final Project — Summer 2016

Team: Angela's Charlies

Approach

Initially, the problem of predicting future locations without position updates seems to depend critically on developing an accurate motion model. However, the motion model can be ignored if, rather than explicitly modeling the dynamics of the hexbug, we instead use a histogram filter to estimate the probability density function (PDF) of all possible future locations conditioned on the final observed state. We discretized the robot domain as shown in Figure 1, then used K-nearest neighbor (KNN) search to reduce bias and generalize the prediction model to unobserved states.

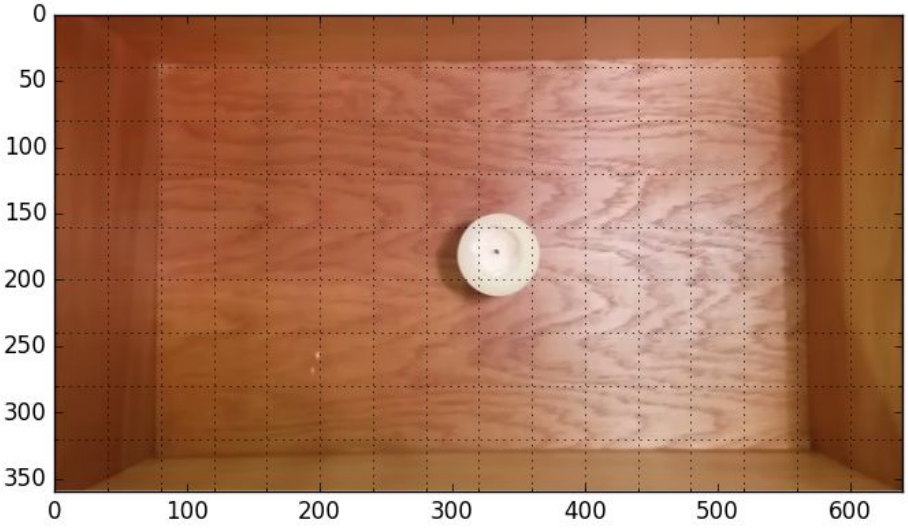


Figure 1: Partitioning of the box into discrete cells for a histogram filter

In some sense, this is an optimal model because it explicitly minimizes the expected value of the L2-norm distance metric, which is the grading criterion. The expected value of the true PDF for the future location conditioned on an appropriate state vector should converge to a point that minimizes the average L2-norm distance from all possible future locations at that point in time. Any other predicted future location would trade smaller errors in some cases with larger errors in others, and the cases that do worse would come to dominate the average because the L2-norm grows quadratically. In practice our model will not be optimal, since the histogram filter only produces a discrete approximation of the underlying continuous problem space when we only have finite data; it would only converge to the optimal solution given an infinite training set and sufficiently large state vector.

Model Overview

Model Parameters

The model is parameterized by the size of the grid cells, the parameters included in the state vector (including the velocity scaling factor), the number of interpolation steps, and the number of neighbors to include in the prediction average. There is also an additional parameter used to partition the training data into learning episodes controlling the gap between the starting frame of each episode.

State Vector

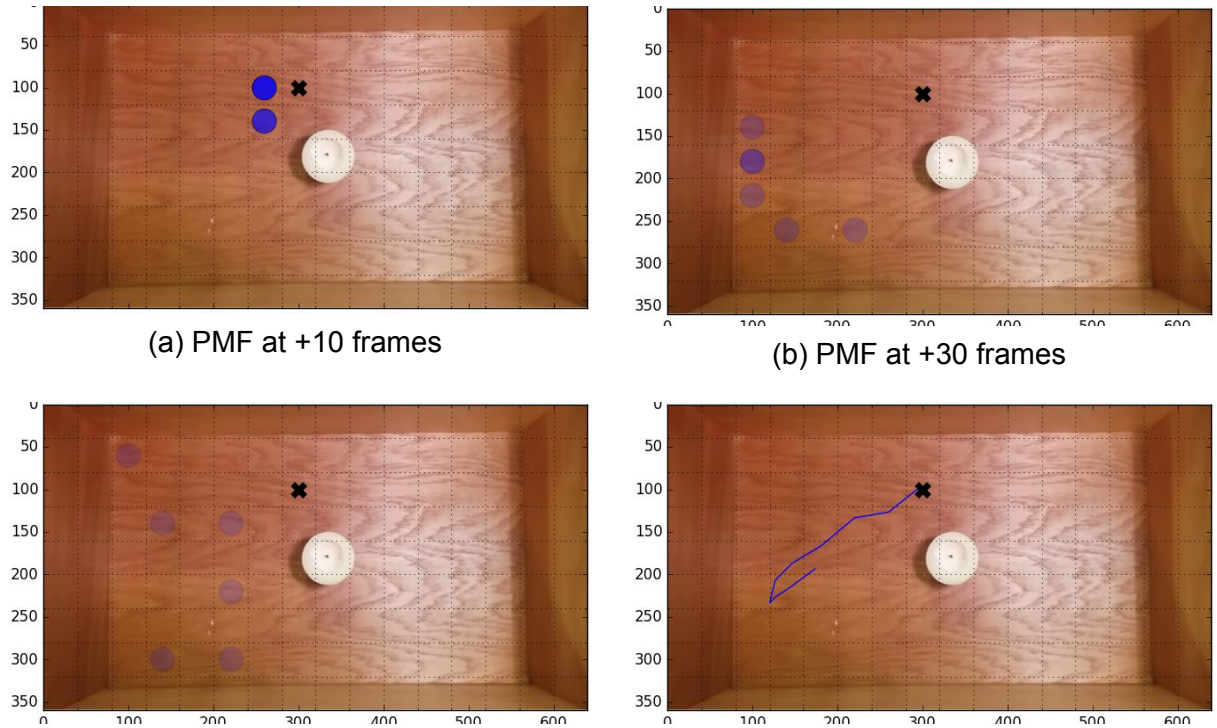
Perhaps the simplest state vector is just the (x, y) position of the hexbug centroid. However, bare position measurements lead to poor predictions in most cases because the probability mass functions (PMFs) are not conditioned on the motion of the bot between frames, so the average prediction for the k-nearest traces tends to approximate the baseline prediction of staying still. However, adding an estimate of the velocity in the state vector improves the predictions because bot motion for the near future tends to be biased towards the observed direction of motion.

Thus our state model consists of a 4-dimensional vector approximating the current location and scaled velocity of the bot. The position is approximated by the grid cell indices of the most recent (x, y) measurement, and the velocity is determined from the backwards finite difference approximation over the last n frames preceding the measurement. The velocity scaling factor, c, is used both to discretize the velocity levels and to control the scale of the distance measurement between states for the KNN search.

$$v_x(t) = c\frac{x(t)-x(t-n)}{n}$$
$$v_y(t) = c\frac{y(t)-y(t-n)}{n}$$
$$S = [x, y, v_x, v_y]^T$$

Making Predictions

A histogram filter is appropriate for this problem because the underlying domain is continuous (albeit with discrete position measurements). The PMFs can be calculated from a simple frequency count within each bin of a discrete grid spanning the domain. We use an array of histogram filters, each describing a PMF (approximating the underlying PDF of the continuous domain) of the future location conditioned on the last observed state for a specific future time, equally spaced through the 60-frame prediction period. The expected value of each PMF is the optimal prediction point for the corresponding future frame. The trace connecting these points through the entire array represents the best prediction based on the training data. Figure 2 illustrates the approximate PMF from a notional starting state along with the interpolated average trace recorded for that state.



(a) PMF at +10 frames

(b) PMF at +30 frames

(c) PMF at +60 frames

(d) Interpolated trace for frames 1-60

Figure 2: PMFs are approximated by frequency counts in each location bin - the black X represents the last observed location, and the intensity of the blue dots correspond to the proportional frequency counts in each cell.

Path predictions $P(t|S_p)$ at frame t are calculated by interpolating between the weighted average of the traces T_i for each of the K-nearest states. Using KNN search helps correct for bias in the PMF approximations due to finite training data and generalizes the prediction model to unobserved states. The interpolation weights w_i are calculated from the normalized euclidean distance between each nearest state vector S_i and the prediction state S_p. Initialization error is reduced by beginning each interpolation trace from the last measured (x, y) position. An example of the k-nearest traces and the returned weighted average path for a notional prediction state are shown in Figure 3.

$$d_i = \sqrt{(S_i - S_p)^2} = \sqrt{(x_i - x_p)^2 + (y_i - y_p)^2 + (v_{x,i} - v_{x,p})^2 + (v_{y,i} - v_{y,p})^2}$$

$$w_i = 1 - \frac{d_i}{\sum_{j=1}^{k} d_j}$$

$$P(t|S_p) = \sum_{i=1}^{k} w_i T_i$$

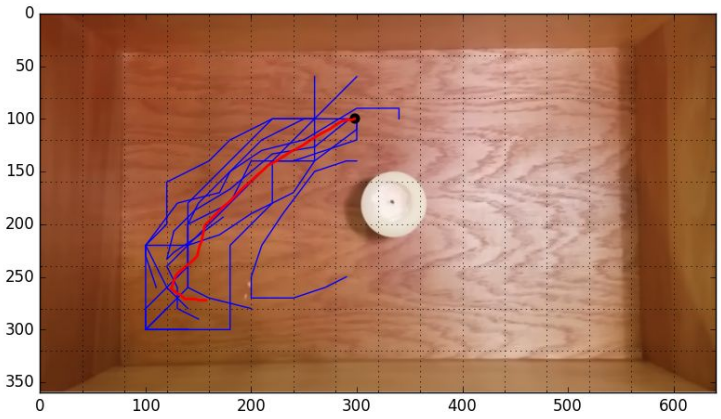


Figure 3: Example prediction (red) computed from the average of the K-nearest expected paths learned from the training data (blue) for the last observed state (black dot)

Data Preprocessing

The model is trained using episodes extracted from the combined training and test data at regular intervals as shown in Figure 4. Each episode consists of an initial sequence of measurements to initialize the state estimate, followed by a sequence of true position measurements. The data is partitioned into training episodes by selecting a random offset from a uniform distribution, then extracting an episode at regular intervals through the remaining data. There may be overlap between the episodes; small overlaps limit the PMFs (since there are fewer observations to train from) while large overlaps may lead to overfitting the model to the training data (especially when sequential episodes are strongly correlated).

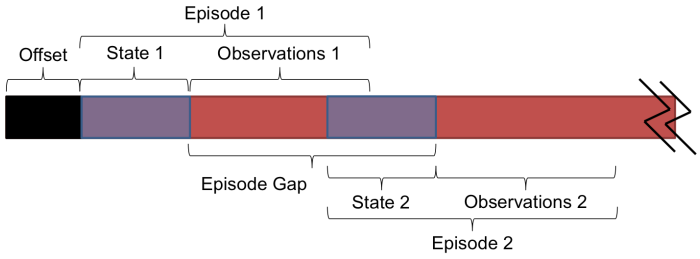


Figure 4: Training data is partitioned into regularly spaced training episodes parameterized by the gap between the last observation of each state predictor. (Note: not to scale)

Results

The predictions for the 10 test cases are shown in Figure 5. Test cases 7 and 8 are particularly interesting. In test case 7, augmenting the training data with the test video yielded a prediction close to the last observed location, demonstrating that the model incorporates motion dynamics without explicitly modeling the bot. In contrast, test case 8 shows that a simple state vector is not always adequate, since there were so many possible outcomes that it essentially returns a null result.

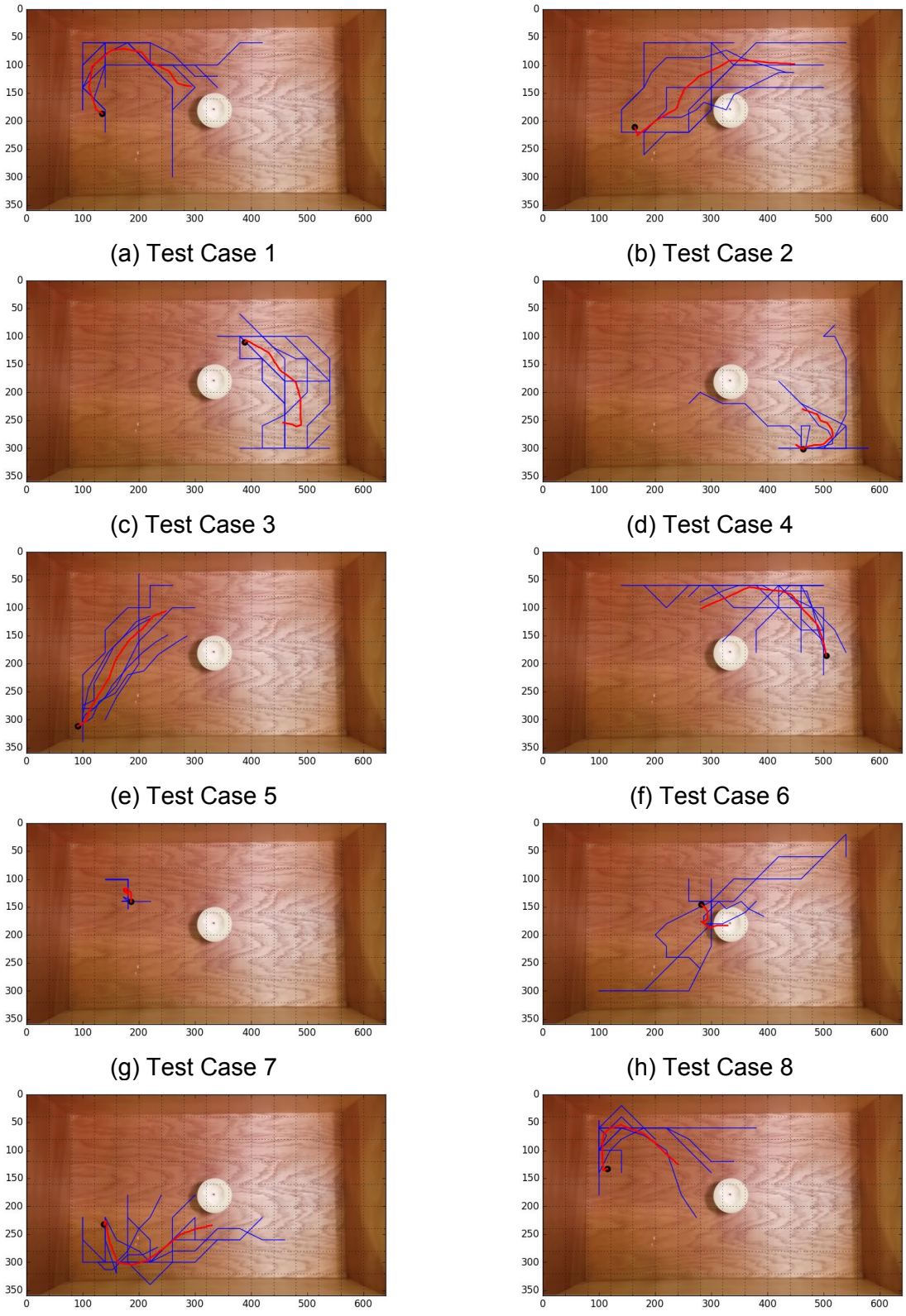


Figure 5: Final predictions for each of the ten test cases, test01 through test10 in (a) through (j)

Table 1 shows the estimated results using our final parameters: 16x9 grid, velocity estimated over 5 frames, velocity scaling factor $c = 7$, 7 nearest neighbors, 6 frame interpolation, and 13 frame gap between training episodes. Results were obtained by performing k-fold cross-validation. All available data (textXX.txt and training_data.txt) was pooled, episodes were generated and divided into folds, and error was averaged across each training/test split. For the N=10 case, k was determined from the size of the training data set to yield cross validation groups with exactly 10 episodes for each run. In the k=5 case, 20% of the data in each run was reserved as a test set while the model was trained on the remaining 80%.

The estimate for $N = 10$ test cases reports the expected value of the average L2-norm error (dropping lowest & highest score) for 10 randomly selected test cases, averaged over many random splits of the training set. This is intended to model the “real” testing scenario. The estimate for $k = 5$ reports the true average L2-norm error for the model (i.e., averaged over large values of N). The “Test” line reports the average L2-norm error (dropping lowest & highest score) for the last 60 frames of the 10 test case videos. The higher error in the middle test case may be due to losing more of the training data to the test set partition.

Table 1: Estimates of average error for various test conditions comparing path predictions from the model to the baseline error.

	Prediction		**Baseline**	
	Avg	σ^2	Avg	σ^2
L2-norm (N=10)	698.5	96.1	1239.5	146.3
L2-norm (k=5)	717.0	9.6	1213.5	23.1
L2-norm (Test)	673.8	166.2	1261.7	262.5

Discussion

Using a histogram filter simplifies the problem by abstracting several issues that complicate alternative approaches. For example, a Kalman filter requires a process model that describes the motion of the hexbug; forecasting the location using a Kalman filter or particle filter requires a motion model with enough sophistication to handle interactions with obstacles in the domain; and, in general, predicting specific trajectories requires information about the location of walls and obstacles within the video. By comparison, a histogram filter avoids explicitly modeling any of those elements because their combined effects are already incorporated indirectly by the observed frequency information that is used to estimate the conditional probability distributions of future locations. The histogram filter naturally handles unreachable areas near the image borders and candle interior without any special cases to handle obstacles, because the bot will never visit those locations in the training data, so the frequency count for those bins will always be zero.

An important nuance of our approach is that we are not attempting to predict the path of the bot per se, but rather attempting to predict a path that minimizes the expected value of the L2-norm error compared to all possible future paths that the bot may travel. In other words, the model incorporates growing uncertainty over time after we stop receiving measurement updates.

We tuned the model parameters using k-fold cross validation (under reasonable assumptions about test case independence, and that randomly selected test cases accurately represent the true distribution) to optimize the expected value of the test statistic. We found that small episode offsets, large grid cells, small interpolation steps, and a wider range of velocity estimates produced the best expected score.

Avoiding Overfitting

One of the main limitations of using a histogram filter is that the estimated probability distributions are entirely dependent on the observed training data; i.e., the model contains no direct estimates of the behavior for states that are not observed during training, and observations are not explicitly generalized between symmetric states. Thus, there is a good chance of the model overfitting the data, and it is difficult to generalize the results to unobserved states.

In this case, the predictions for each starting state quickly overfit the available training data because even coarse state estimates result in a large state space, so most states will never be directly observed with the limited training data available. For example, a state vector consisting of xy position on a 16x9 grid and velocity discretized in 8 bins for X and Y directions yields a state space with 9,216 values, while partitioning the training & test video data with starting states every 5 frames only yields approximately 10,000 episodes.

We evaluated overfitting in our model using learning curves computed from the average over k-fold cross validation (k=5), as shown in Figure 6. The learning curve shows some indication of overfitting as the training and cross validation curves diverge, but both errors are continuing to fall, which suggests that the model may still be improved by continuing to add more training data. It is likely that providing more training data and increasing the gap between episodes (to ensure independence between the training and cross validation sets) would result in better overall performance from the model.

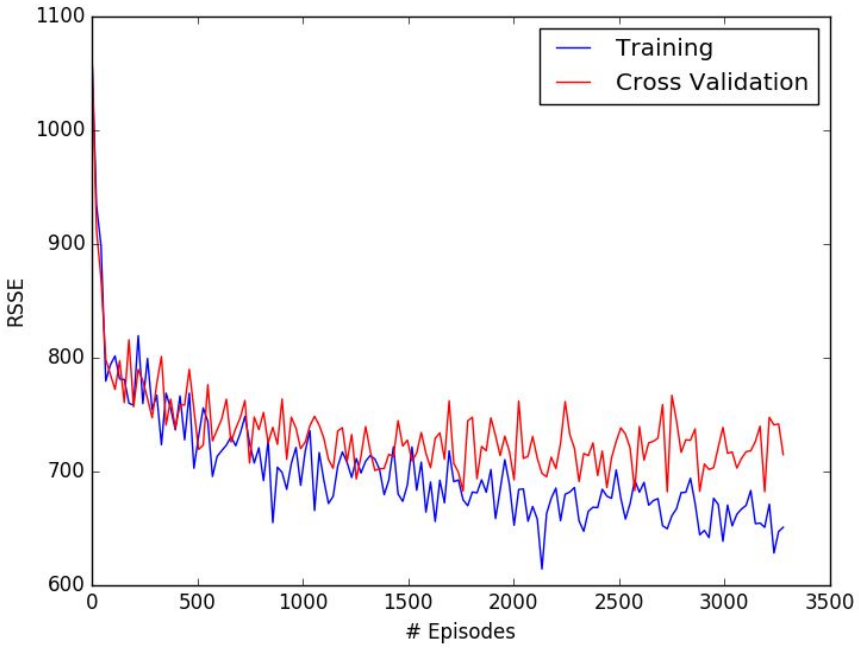


Figure 6: Training curve estimating average prediction error as a function of the number of training episodes incorporated in the PMF estimates

Alternative Approaches & Improvements

Processing Raw Video Data

Early on in the project, an attempt was made to process the raw video data to yield better state estimations than those provided in the test files. Several attempts to process the data were attempted, including using filters to find the robot and estimate the centroid and using OpenCV to find the object in the image. Initial tests of small subsets of the data yielded similar results to the given data. Adding this additional layer of complexity did not appear to generate significantly better data and was scrapped. Better processing techniques could still enable manual processing of the raw video data to produce more accurate or subsampled data, which could improve the overall algorithm.

Avoiding Interpolation

Instead of interpolating, we could directly calculate the expected position for every frame. However, we must keep a coarse discretization of the state space with such limited training data, which would cause many sequential measurements to fall in the same bins such that the optimal prediction would not move much until the bot moves far enough between frames to fall in another bin. In contrast, selecting a large enough gap to ensure a change in the expected value of the approximated PMF and interpolating between the points smoothes out our predictions. With more training data, the interpolation could be dropped because the grid size could be made smaller and the PMF would converge towards the true distribution.

Comparison With Other Prediction Models

The Kalman filter and its variants may not be optimal in this application unless the distribution of the prediction and measurement noise are Gaussian. If it is not Gaussian, a particle filter may work better, but its accuracy is contingent upon a motion model that accurately represents the hexbug movement, otherwise the random variations of the particles will not capture the probability distribution of future locations.

To explore the possibilities for motion model-based approaches, we attempted to hand-tailor a motion model for the Hexbug. The initial model called for the bug to deflect in the direction with lower velocity whenever it encountered an obstacle; if no obstacle was encountered it would continue in a straight line along its current heading. Velocities were calculated for all robot positions p(t) in the training data using the robot's position p(t-1) measured at the previous timestep.

A simplified world model was built by noting which locations the Hexbug visited; (x,y) coordinate pairs that the Hexbug never approached within 10 pixels during any of the training runs were assumed to be physically inaccessible and treated as obstructions. A collision was then flagged and deflection occurred whenever the robot's velocity at time t would be expected to carry it into an inaccessible location at time t+1. Initial guesses were straightforward to populate based on our underlying assumptions about the physical laws of the universe:

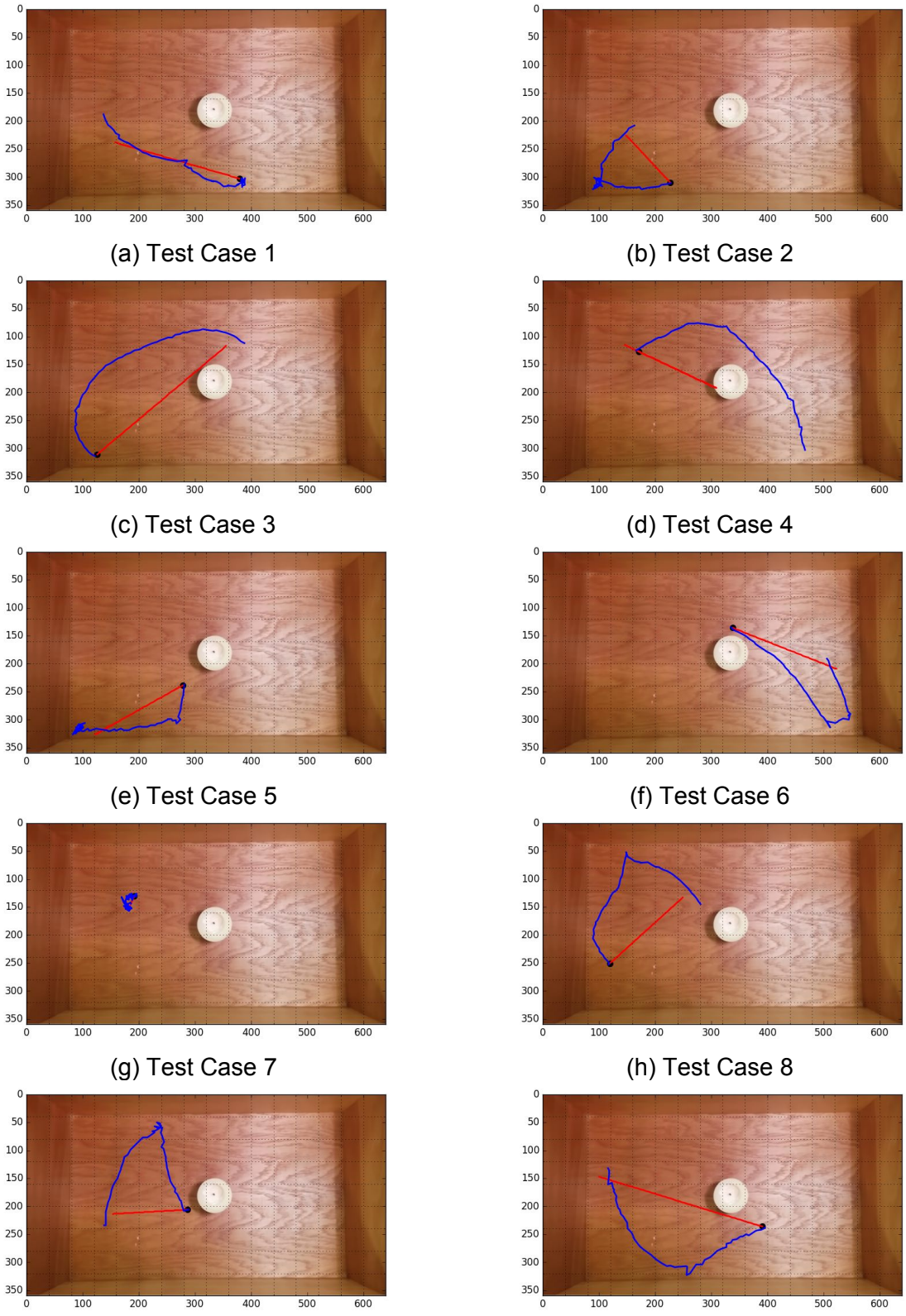
$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 7: Initial guess for prediction matrix for collision/noncollision model estimation. The four columns and rows represent x position, y position, x velocity, and y velocity

As seen above, we expect that the robot's position at time t+1 will always be a linear combination of its current position and its velocity, but it remains to determine how the robot's velocity evolves over time.

Table 2: Estimates of average L2-norm error comparing path predictions from the explicit motion model to the baseline error.

	Model Prediction		**Baseline**	
	Avg	σ^2	Avg	σ^2
L2-norm (N=10)	1174.1	147.9	1239.5	146.3
L2-norm (Test)	583.1	145.5	1261.7	262.5



(a) Test Case 1

(b) Test Case 2

(c) Test Case 3

(d) Test Case 4

(e) Test Case 5

(f) Test Case 6

(g) Test Case 7

(h) Test Case 8

(i) Test Case 9

(j) Test Case 10

Figure 8: Predicted paths for the final 60 frames of the ten test cases, test01 through test10 in (a) through (j)

The overall average from random testing showed that this model performed poorly compared to the histogram filter, and only marginally better than baseline. However, interestingly, it performed better than the histogram filter on the predictions for the last 60 frames of the ten test videos. This may indicate that the test videos differ significantly from a uniform random test sequence.

We also experimented with a model that attempts to account for some of the statistical evidence could yield better results. To explore this initial intuition, we constructed an alternative estimator which relied on two prediction matrices - one intended to model robot velocity immediately after a collision, and one intended to model robot velocity when no collision occurred. Once training datapoints were separated into collision and non-collision sets, velocity change was measured at each timestep. A least-squares fit was used to determine the velocity relations in the post-collision and non-collision filters.

$$\begin{bmatrix} 1.0011 & -0.0018 & 0.4485 & -0.0084 \\ -0 & 1.0011 & 0.0147 & 0.4409 \\ 0.0011 & -0.0018 & 0.4485 & -0.0084 \\ -0 & 0.0011 & 0.0147 & 0.4409 \end{bmatrix} \begin{bmatrix} 0.9992 & 0.0013 & 0.3002 & 0.1216 \\ 0.0011 & 0.9973 & 0.0939 & 0.4080 \\ -0.0008 & 0.0013 & 0.3002 & 0.1216 \\ 0.0011 & -0.0027 & 0.0939 & 0.4080 \end{bmatrix}$$

Figure 9: Deduced motion models which minimize L2 error

These matrices were then used for predicting the test datapoints: the normal (non-collision) calculation was performed first, and the alternative (post-collision) values used whenever the normal (non-collision) filter values would carry the robot into an inaccessible location.

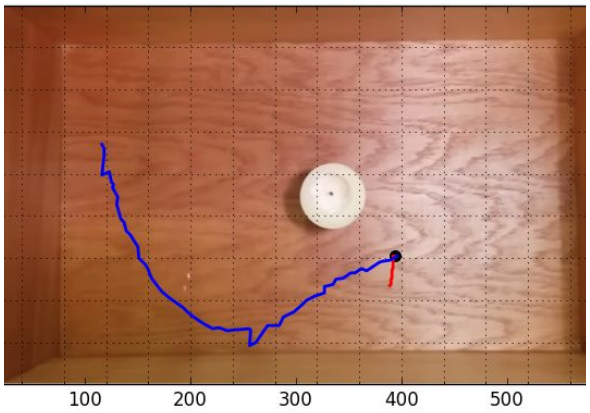


Figure 10: A representative trace from the deduced motion model

This bot's rapid slowdown (note that both velocity diagonals are << 1.0) supports our initial intuition: linear motion models are inadequate for this scenario, because any predictor which does not (implicitly or explicitly) incorporate nonsymmetric location and heading information will perform poorly, tending to average out over the entire world given enough data.

Furthermore, given the lack of topological data about the box floor, our *only* way of modeling it is by making inferences about its effects from observations of the bug. This strongly suggests that our kNN approach is close to optimal given the right parameter selection, since it captures information about location and velocity changes based on this physical proximity.

Conclusion

A histogram filter provides a reasonable estimate for the conditional probabilities of future location based on the training data, and produces a good estimator for the expected path to minimize average error according to the L2-norm distance metric. Averaging the predictions using KNN search further reduced the impact of overfitting, and generalized the prediction model to unobserved states to compensate for limited training data. Cross validation testing and training curves indicate that the model is robust, and that overfitting is not a significant factor in our predictions.