Estimating the State of Robot with Dynamic Bayesian Network

Zhuofu Bai

Abstract

The current research on the safety of robotic systems requires accurate estimation of critical system states and parameters. In order to estimate robot's state, I trained a Dynamic Bayesian Network implemented in MATLAB and use an decision tree to improve the accuracy of estimation. I hypothesized that the conditional probability distribution is Gaussian. To evaluate the performance of the DBN, I trained the algorithms on the sampled data from 14 different trajectory and attempted to use the DBN model to track and generate the novel trajectories. From these experiment, I found that DBN out-performs simple linear Gaussian model in Robot state estimation when the training set is large enough and representative to the whole population. But some additional work is necessary in the future to improve the accuracy of estimation.

1.Introduction

One of the biggest challenge in Robotics is to create systems capable of operating efficiently and safely in natural, populated environments. To address these challenges, a lot of work has been done by researchers [2] [3]. In this project, I developed a Framework with Dynamic Bayesian Network and Regression tree to estimate the state of a biopsy robot. The rest of the paper is organized as follows: section 2 describes the robot and the robot's parameters. section3

2. Robot and State data

In this project, our robotic system is a 5 DOF small animal biopsy robot, which is a robotic systems for image-guidedneedle-based interventions on small animals(fig.1)[4]. The State of Robot is defined as the values of all parameters of the Robot at a specific time t. In our project, when the Robot moving its needle the Robot state at time t is represented as a 34 dimension feature vector, which includes the reference needle position, actual needle position, reference needle direction, actual needle direction and parameters of each degree of freedom.

We call the feature vector which represents the state of Robot as state data. The experiment dataset in this project is the state data collected at different time t.

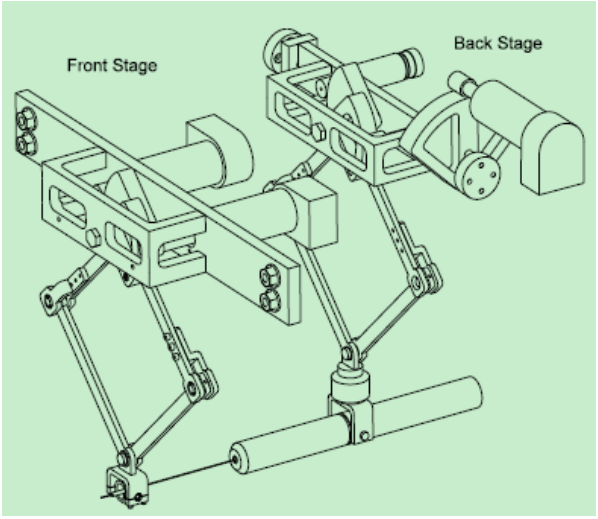

Figure 1

The goal of our project is to train a machine with a set of state data sampled from different trajectory. After training, given the data of the Robot's state at time t in arbitrary trajectory, the machine can estimate the state of the Robot at time t+1. This course project is part of the research project of robot safety, so among all the Robot's parameters, we are most interested in the actual needle position. Fig.2 is the reference trajectory of the needle when the robot did an insertion. Due to the existence of noise, in the real environment, the actual trajectory of the needle cannot be as smooth as fig.2. Thus, estimating the position of the needle in real environment is necessary for Robot safety. In this project, we did a lot of experiments to tracking and estimate the needle position under the noise data.

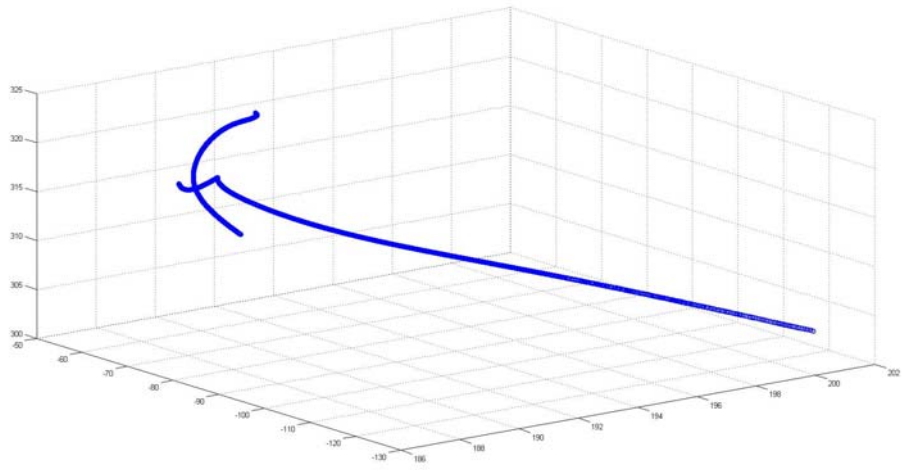

Figure 2

3.Dynamic Bayesian Network

a. Definition

A dynamic Bayesian network, or DBN, is a Bayesian network that represents a temporal probability model of the kind described by the fig. 3

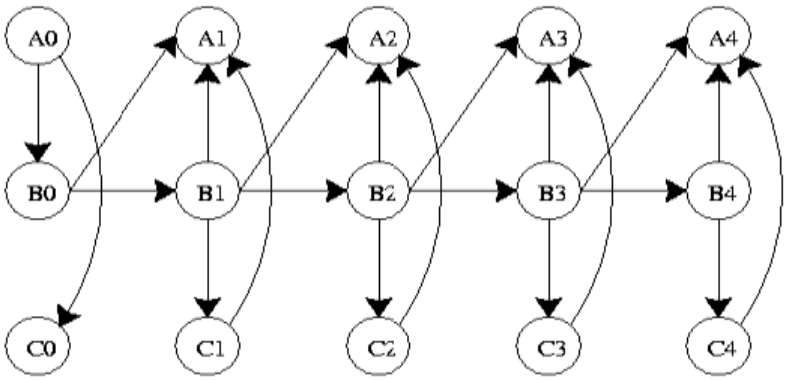

Figure3

In fig.3, Ai, Bi and Ci are the state parameters at time i. Every time slice of DBN is a Bayesian Network. The DBN composed by two Bayesian Networks was called TBN. DBN can be viewed as rolling a TBN from time t=0 to t=n. Typically, a DBN has the following semantics:

a. First-order Markov assumption: the parents of a node can only be in the same time slice or the previous time slice, i.e., arcs do not across slices.
b. Inter-slice arcs are all from left to right, reflecting the time.
c. Intra-slice arcs can be arbitrary as long as the DBN is a DAG
d. Time-invariant assumption: the parameters DBN do not change over time [1].
e. The resulting joint probability distribution is then defined by

$$P(X_t \mid X_{t-1}) = \prod_{i=1}^{N} P(X_t^i \mid Pa(X_t^i))$$

Where, X_t^i is a node at time slice t, it can be a hidden node and an observation node. $Pa(X_t^i)$ are parent nodes of X_t^i, they can be at either time slice t or t-1.

b. Parameter Learning

For DBN, the parameter learning has two methods:
Offline learningParameters must be tied across time-slices. The initial state of the dynamic system can be learned independently of the transition

matrix.Online learningAdd the parameters to the state space and then do online inference (filtering).

In this project, we mainly focus on offline learning, because we did not update the model in testing step. We do not know the dependence relationship between attributes, so we set the slice t is fully connected to slice t+△t in DBN (fig.4). If the one attribute did not depend on another attribute, the probability dependence will be very small after training.

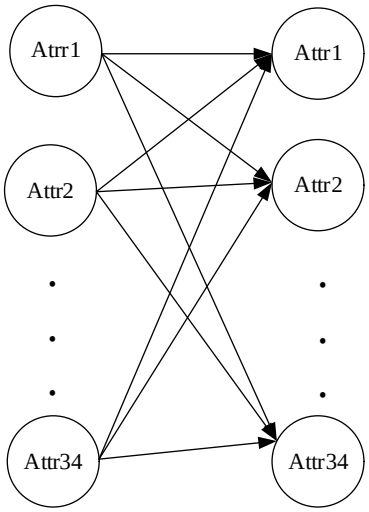

Figure 4

We use linear Gaussian model to model the probability dependence between two slices. With linear Gaussian model, we can use a linear regression to estimate the value of arbitrary attribute at time t+△t. That is, we assume the noise is Gaussian and the relationship between attributes is linear:

$$a_n = b_0 a_0 + b_1 a_1 + \cdots + b_n a_n$$

Suppose we have trained a linear regression to fit the training data. When a novel state data comes, the estimated value at t+1 can be calculated in two ways. First, just use the linear regression to estimate the value of the state at t+1. Second, use the result of linear regression μ as the expectation of the state at t+1, and then randomly sample a value from the Gaussian model $N(\mu,\sigma)$ as the estimation, where σ is the standard deviation of the added noise.

In the next step, we use regression tree to improve the accuracy of the DBN in state estimation .

4.Regression Tree

To introduce Regression Tree, we start with a simple example. Suppose a car was accelerating along axis and a camera record the car's position every. The function of was plotted in fig.5. If we want to model the data sampled from, the most straight forward way is to use linear regression. However, the function is not a linear function, so a linear regression may leads to larger errors. A better way to model the data is to split the data set into two parts, ex. fig.6. For each subset, use a linear regression to model the data. Similarly, we can recursively use this method to split and model each subset. In this way, we build a binary tree to separate the data space. Each node of the tree represents a separate of the data space. At each leaf node, there is a linear regression to model the data.

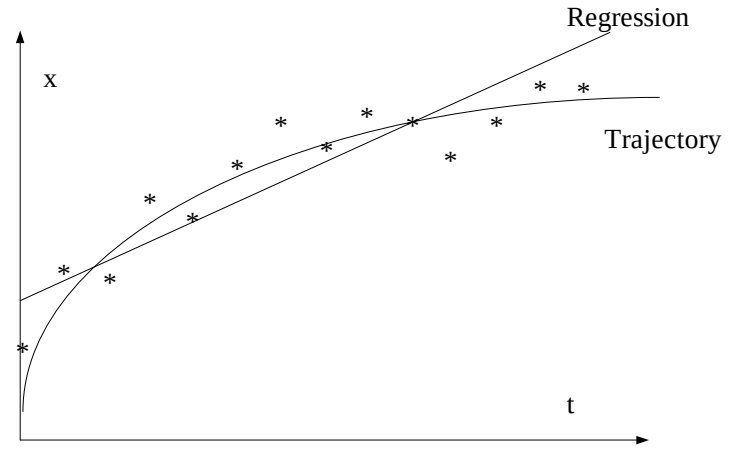

Figure5

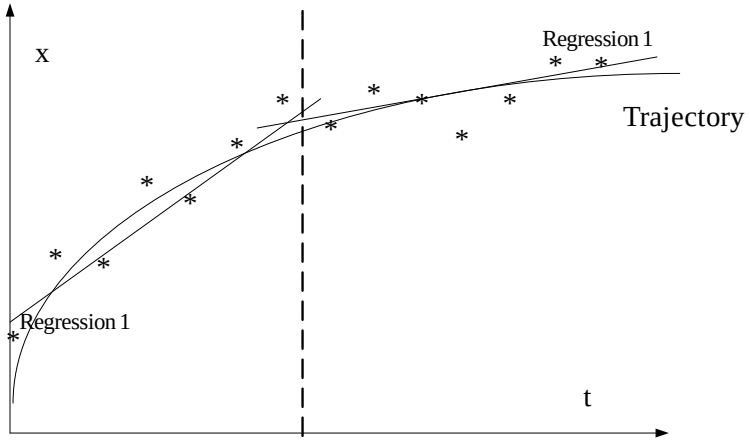

Figure6

The algorithm chooses the split that partitions the data into two parts such that it minimizes the sum of the squared loss from the mean in the separate parts. The algorithm of building a regression tree is similar as the algorithm of building a decision tree for the data set with continuous attribute. The only difference is that the split point chosen is based on information gain.

When the training data has multiple attributes, then every value in each attributes is a candidate split point. For example, if the training size is m and each training data has n attributes, then there are m*n split candidates. An efficient algorithm to build a regression tree is that randomly sampled proportion of split candidates, then choose the split point from the sampled candidate. In this way, the regression tree is sub-optimum.

Intuitively speaking, a regression tree will outperform a linear regression for modeling a nonlinear dataset. But there is a potential problem in building the regression tree. Suppose every the training data has distinct continuous value. If we build a full regression tree for the data set, each leaf node of the tree would contain only one data, which can be perfectly modeled by a linear regression. This means the regression tree is over-fitting the training data, resulting in poor performance on real life data. Therefore, the tree building process should stop when each node reaches a user-specified minimum node size and becomes a terminal node. An alternative way is to set up a threshold for the mean square loss. If the mean square loss of the current node is less than the threshold, set the current node as the terminal node. Generally speaking, a good regression tree required larger size of training data to make sure each leaf node got enough training data for linear regression. This leads to more computation cost and running time comparing to train a single linear regression for the whole data. In this project, we set the minimum node size to the 3n for the data set with n attributes.

Another function of regression tree is to pruning the attributes, when the training data has multiple attributes. If the linear regression of a leaf node results in small square loss, then we can retraining the linear regression with only the attributes in the path from root to this leaf node. Regression tree may be considered as a variant of decision trees, designed to approximate real-valued functions instead of being used for classification tasks. Because regression tree often need a lot of time for training, we limit the tree height to 3 level in this project.

5.Experiment

As mentioned previously, I evaluated the performance of DBN on then data

from Matlab simulator. Train a DBN with all the 34 attributes requires a high computation cost, so for the course project purpose, I first use 6 attributes to estimate the need position. The 6 attributes are:
Needle Position: X, Y, Z
Direction of the Movement: RX, RY, RZ

Here we use PX, PY, PZ to denote the actual needle position along X axis, Y axis and Z axis respectively. In the training phase, the train set contains 6000 examples randomly sampled from 14 different trajectories. All the training data was added Gaussian Noise. Next, A 3 level regression tree was built for PX, PZ, RX, RY and RZ respectively. For PY, we increase the height of the regression tree to 4, because in the experiment we found that in most trajectories, PY has larger variance than that of PX and PZ. In the regression tree, the minimum node size is 21.

For comparison, we use a single linear Gaussian model to fit all the training data. And use this linear Gaussian model as the base line in the experiments. The mean residual square of DBN and Base line are listed in Table 1.

Table 1

Mean Residual Square	X	Y	Z
DBN	1.7891	7.0827	4.8960
Base Line(Simple Gaussian)	12.3634	11.1598	20.025

a. Tracking

The first experiment is to use the trained DBN model to track the needle position of the robot. The DBN was trained by the data with added Gaussian noise. Here, the tracking is different from that of Kalman Filter, because we do not update the data model when we got a new observation. To test the tracking performance of DBN model, we use two different kinds of data set. One is the noise free data. The other is the noise data. The time interval $\triangle t$ is set to 0.1s.

The tracking experiment result with noise data is shown in fig.1. In fig.7, the blue dot is the true Robotic needle position and the red dot is the estimated position. Because the testing data is added Gaussian noise, the trajectory cannot be recognized clearly from the figure. From fig.7, we can see most of the estimated position is close to the true position. Also, we can see most of the estimated position is on the left side of the corresponded true position. This indicates the error is mainly on X axis, which means more training is needed for PX data.

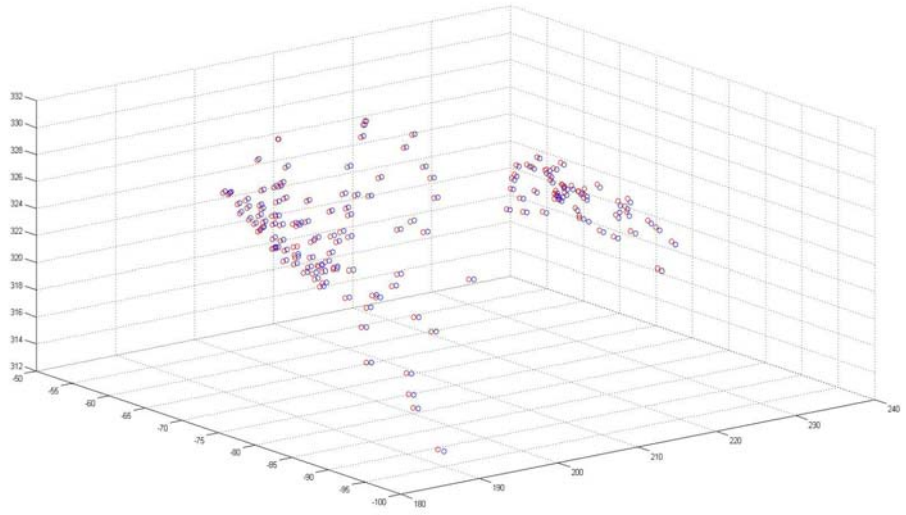

Figure 7

Next, we use the same DBN to test the noise-free data. The result was shown in fig.8. In fig.8, the start point of the trajectory is (194, 73, 325). From fig.8 we can see DBN has a good performance on noise-free data, but most of the estimated position is at the up left side of the true position. This means the model of PX and PY need to be improved. We also use a simple linear Gaussian model to tracking the same trajectory for comparison. The result was shown in fig.9. By comparing fig.82 and fig.9, we can see at the beginning of the tracking, simple linear Gaussian performs better than DBN. But in fig.8, the distance between estimated position and true position reduced when tracking time increased. In contrast, in fig.9, the distance between estimated position and true position increased while the tracking time increased.

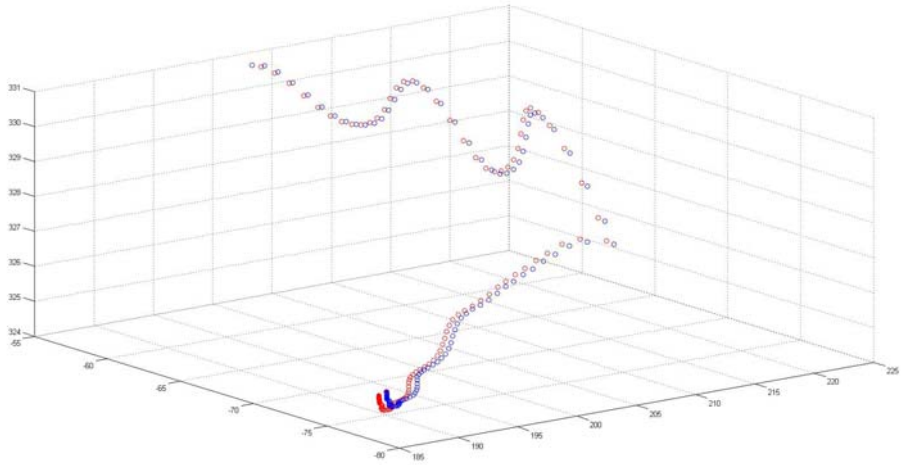

Figure 8

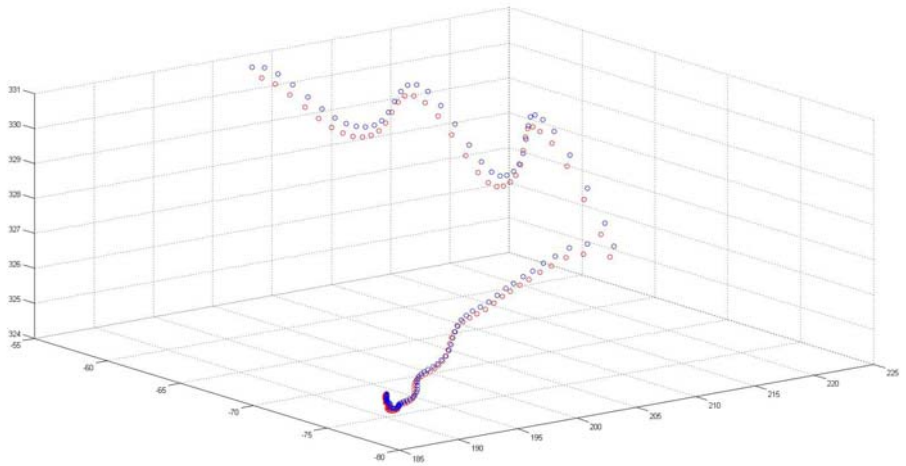

Figure 9

Fig. 10 shows a tacking failure with DBN, the estimated PX value is suddenly reduced by 20, when PZ value is less than 324. This is because when the PZ value is less than 324, the PX value will be estimated by a different linear regression. To overcome this problem, there are two possible ways. One is increase the height of the tree. That is, we need to use more linear model to fit the dataset under the condition PZ<324. Another way is to make the DBN working as a Kalman filter. When a new observation was received, the DBN update its model.

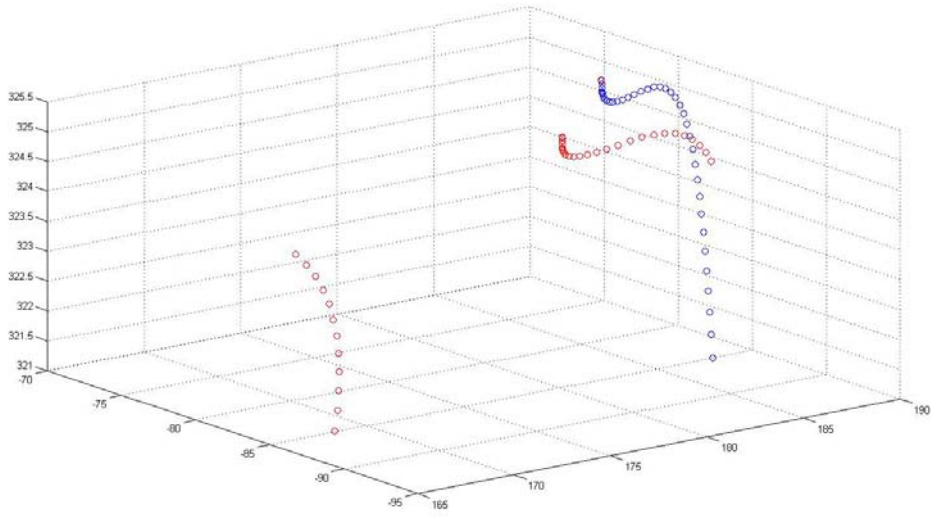

Figure 10

b. Generating Trajectory Automatically

Another experiment is more interesting. We use the DBN to automatically generate the trajectory. The trajectory was generated by the following steps:

1. Input the state data at time t0
2. Estimate the state data at time t1 by DBN.
3. Input the estimated state in to DBN model for estimating the robot state at time t2.
4. Repeat step 2 and 3 to generate a trajectory.

The time interval of training set and testing set are both set to 0.01, because in trajectory generation, the error will be accumulated when time is increasing. The training set with time interval $\triangle t=0.1$ has larger training error than that of the training set with time interval $\triangle t=0.01$. Thus, if we use $\triangle t=0.1$ instead of $\triangle t=0.01$ in trajectory generating, it is high possible that the generated trajectory is deviate far from the true trajectory.

The test data in this experiment is also noise-free data. Fig.11 shows the trajectory generated by DBN. From fig. 11, we can see the DBN performs well on estimate PX and PZ. In estimating the value of PY, the DBN decrease the value of PY when time increase. However, in the actual trajectory, the value of PY is monotone increasing. This means the DBN fails estimating the value of PY at the beginning.

The trajectory generated by simple linear Gaussian model was shown in

fig.12. In fig.12, we can see the simple linear regression estimate the value of PY better than DBN, but fail in estimating PX. The result of fig.11 and fig.12 indicates that for some attributes has large variance, it is easy to get bias training data. To solve this problem, the simplest way is to use simple linear Gaussian model to estimate PY and use DBN to estimate the value of PX and PZ. But from fig. 12, we can see the error of estimation is still considerable. Another way is to use more fancy sampling algorithm to get the training data more representative to the whole population.

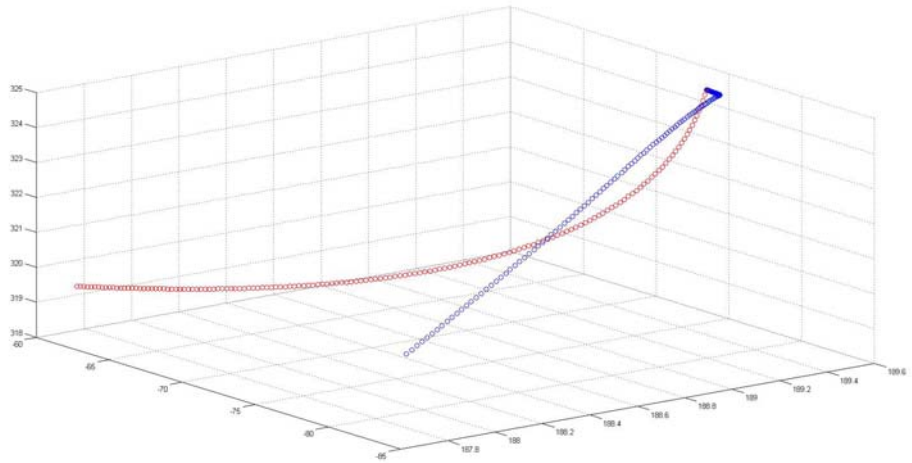

Figure 11

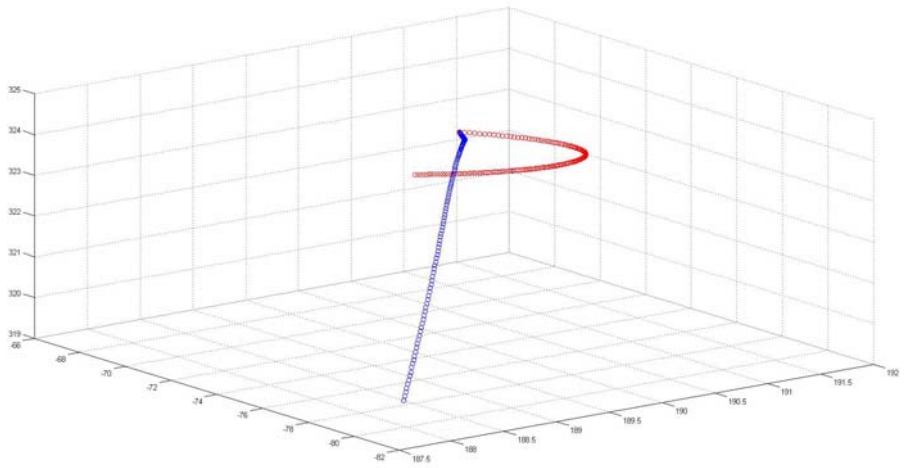

Figure 12

The last experiment is to build a regression tree with all the 34 attributes to estimate the Needle Position. The computation cost of

building a regression with 34 attributes is very high, so we only trained one model to estimate the X value (PX) of Needle Position. To train this model, the training set has 10000 examples sampled from 14 trajectories.

The range of PX value in the training set is [130 260]. The mean residual square is 46.5910. Comparing with the result in table 1, we can see mean Residual Square of the DBN with 6 attributes is smaller than that of the DBN with 34 attributes. This means a 3 level tree is not enough for well estimating a state with 34 attributes, more training is necessary. Fig.13 shows the 3 level regression tree for the state data with 34 attributes.

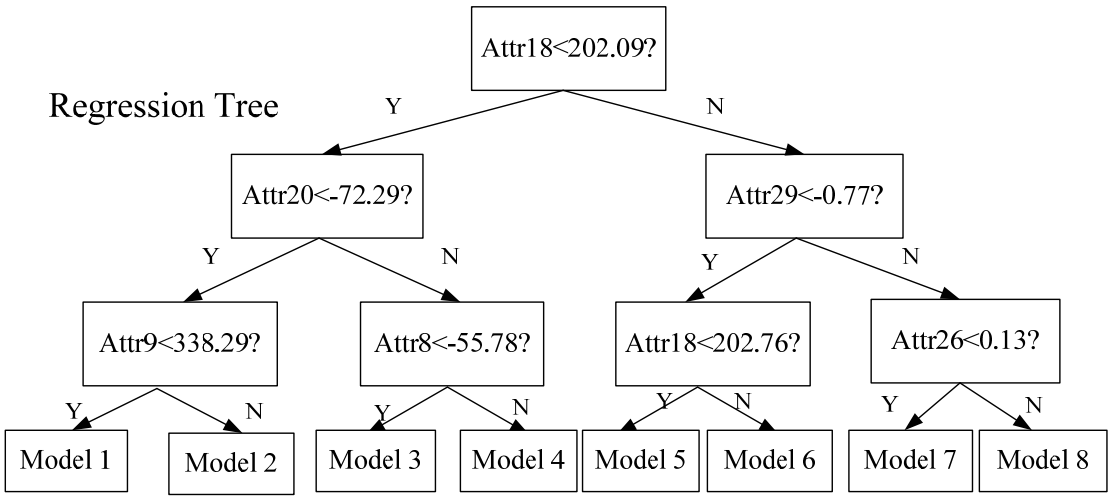

Figure 13

6.Conclusion

In conclusion, I use the Dynamic Bayesian Network with Regression Tree to model the training data with 6 attributes. The trained model was used to tracking the trajectory of the actual needle position and automatically generate trajectory by providing the start point. The result shows DBN has potential to out perform the single linear Gaussian model, when the training data is large enough to build a good regression tree. But to get high accuracy of estimation, the DBN model need to be trained by more representative data and assisted by a regression tree with more levels. Also, to train a DBN with 34 attributes, the dataset has to be at least larger than 10000 to get a accurate estimation.

7.Future Work

In this project, there are still some fields need to be improved in the future. First, in our DBN model, only the inter arc are considered. But some of the robot's parameters are depend on each other, ex. the reference needle position and the actual needle position. Thus, the network within

each slice should be considered in our DBN model. for example, the needle position on time t+Δt not only depend on the parameters at time t, but also depend on the parameters at time t+Δt. Training DBN with 34 attributes requires lot of time for computation. Thus, another future work is that the to tracking the needle position and automatically generating trajectory with DBN of 34 attributes. As mentioned earlier, training the model of PY requires the training data are representative to the whole population. The future work may also need some state of art sampling method to get training data.

Reference

[1] Haipeng Guo, Dynamic Bayesian Networks, KDD Group Seminar
[2] S.Chouraqui, M. Benyettou, State Estimation for Mobile Robot Using Neural Networks.
[3] Dirk Schulz,, Wolfram Burgard, Probabilistic state estimation of dynamic objects with a moving mobile robot.
[4]Ozkan Bebek, Myun Joong Hwang, Baowei Fei and M. Cenk Cavusoglu,
Design of a Small Animal Biopsy Robot