[Appendix]
Deep robot learning from demonstration (D-LfD): suturing with minimally invasive robot

D-RLfD datset

We have collected a dataset of 60 trials of inserting a circular needle through a deformable synthetic soft tissue. The phantom is a synthetic polyurethane sponge cuboid shape (10x5x2 [cm]). The link to the publicly available dataset will be added to the paper once anonymity is no longer a requirement.

We will also release our Deep-Robot Learning from Demonstrations (D-RLfD) implementations on the dataset repository and accept to host (on the repository for benchmarking) other peer-reviewed state-of-the-art (SOTA) D-RLfD implementations using our dataset. So, SOTA in D-RLfD will be easily available to the community for comparison of their novel D-RLfD implementations.

F-CNN

Convolutional Neural Networks (CNNs) are exploited for feature extraction in needle insertion dataset and additional to the well-know architectures we have also tried many customised CNNs to enhance the performance. As such, we have designed a CNN which could yield a better result (regarding MSE presented in Table 1) called F-CNN. The architecture of F-CNN consists of five types of layers: convolutional layer, pooling layer, flatten layer, concatenation layer and fully-connected layer. We used the combination of these layers to construct our CNN model. Note that the depth of the networks and the parameters of the models are chosen by trial-and-error after evaluating validation dataset for more than 300 different CNNs. F-CNN architecture, the best CNN as feature extractor, is presented in Table 2.

The observation o_t is fed into three 3×3 convolutions with stride 1, followed by 2×2 max-pooling layer. This process will be repeated for 5 times, just the filter size of convolution layers are doubled in each process. Eventually, the output of the last convolution layer will be flattened. The current robot action vector and calibration data are processed by feeding into a two layered perceptrons with the size of 15 and 25 for each dense layer. The augmented robot action data will be added to the output of the last convolution layer after being flattened and the concatenated data will be fed into a three layered percpetron network with the size of 80, 20 and 7 for each dense layer, after which the net network outputs the next robot action vector through a linear function. The activation function of all convolution and dense layers is Relu, and the optimization method that is used in our problem is Adam with its default parameters.

Figure 1 shows the general architecture of the discriminative model.

Model	MAE †	AE ‡	Loss	E > 0.01 ‡‡
LeNet	0.5982	0.3845	0.3347	98.57
AlexNet	0.3954	0.2651	0.1478	86.20
VGG19	0.1334	0.0187	0.0201	26.54
ResNet18	0.0984	0.0187	0.0158	27.43
GoogleNet	0.1458	0.0156	0.0130	27.65
F-CNN	**0.0947**	**0.0136**	**0.0137**	**27.03**

† Maximum Absolute Error
‡ Average Error
‡‡ No. of Error elements > 0.01 (%)

Table 1: Evaluation of tested CNN architectures; F-CNN is our customised CNN.

Input – Conv2D – Conv2D – Conv2D – MaxPooling2D – Conv2D – Conv2D – Conv2D – MaxPooling2D – Conv2D – Conv2D – Conv2D – MaxPooling2D – Conv2D – Conv2D – Conv2D – MaxPooling2D – Conv2D – Conv2D – Conv2D – Flatten – Concatenate with calibration Matrix – Dense – Dense – Dense

Table 2: F-CNN (Feature extractor CNN) architecture yielding the best performance as per results in Table1.

Bootstrap Aggregation (Bagging) for Regression:

We know that the observation that we have from a system is coming from a random variable and in most of the cases the true distribution of this random variable is unknown to us. As such, the performance of the models that we train on a collected dataset can have variation coming from the variance of the random variable. Take needle insertion dataset for instance; If we have collected a new dataset on the same setup, the newly collected data have a degree of variation to the dataset that we already have and this would have led to train a slightly different model. Ensemble learning techniques try to evaluate the uncertainty/variation of the performance of

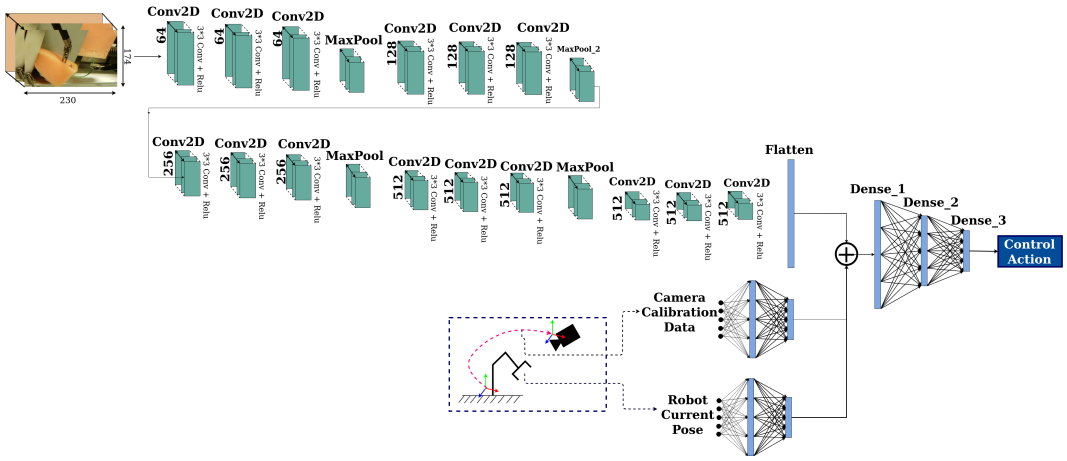
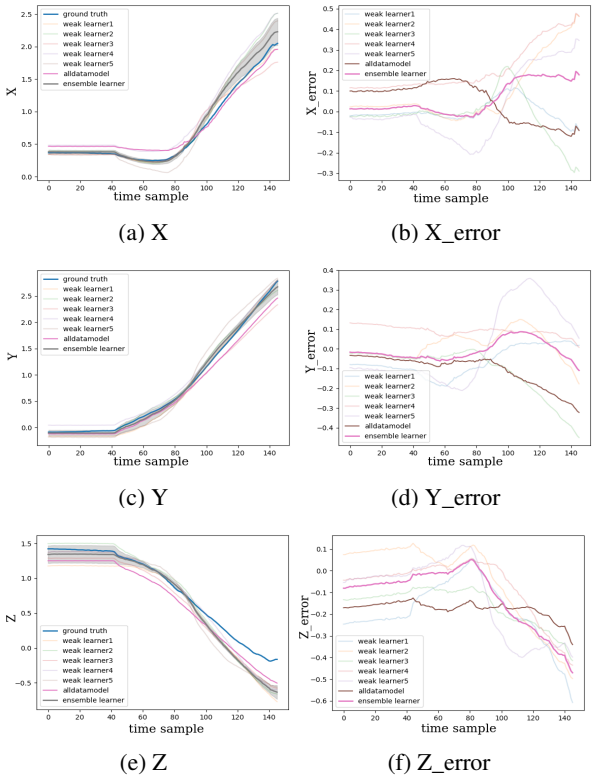


Figure 1: Feed-forward CNN model

the trained model on various unseen test data. Bootstrap Aggregation (Bagging) is an Ensemble Learning method which trains multiple models in parallel on new generated data by bootstrapping and by deterministic averaging computes the output of the ensemble model. This averaging reduces the variance of the ensemble model relative to each of the weak learners which were trained on bootstrapped data.

Bagging for CNNs

Initially we have used Bagging on the F-CNN network which had the best performance and figure 2 shows the result for position outputs and corresponding tracking error values for weak learners and the ensemble model accompanying a CNN model which was trained on all of the original dataset. The gray band shows the 95% confidence interval which is the prediction values added and subtracted by two times of standard deviation. Generally, convolutional networks may not have very good performance in state estimation but here to show how using recurrent networks can enhance the prediction accuracy, in the first step CNNs have been used both for feature extraction and state estimation. As figure 2 shows the ensemble model has fairly good tracking performance on position components of the output. Although some of the weak learner may have poor tracking on some regions but the ensemble model can track the ground truth with a stable performance. The confidence interval bands for position outputs are also narrow enough for a stable tracking. Figure 3 presents the orientation outputs for the models. In our training procedure we have used unit quaternions for orientation representation. However, to have a intuitive result in the paper Euler angles have been reported. There are couple of obvious differences between the graphs in figure 3 and position outputs in figure 2. Tracking performance is relatively poor and that is due to the fact that orientation control is usually more challenging in comparison with position in Robotics. The second difference corresponds to the wider confidence interval for orientation predictions. It can be stated that although CNNs may have fairly acceptable performance on position outputs but for orientation they fail to have good tracking performance.



(a) X (b) X_error

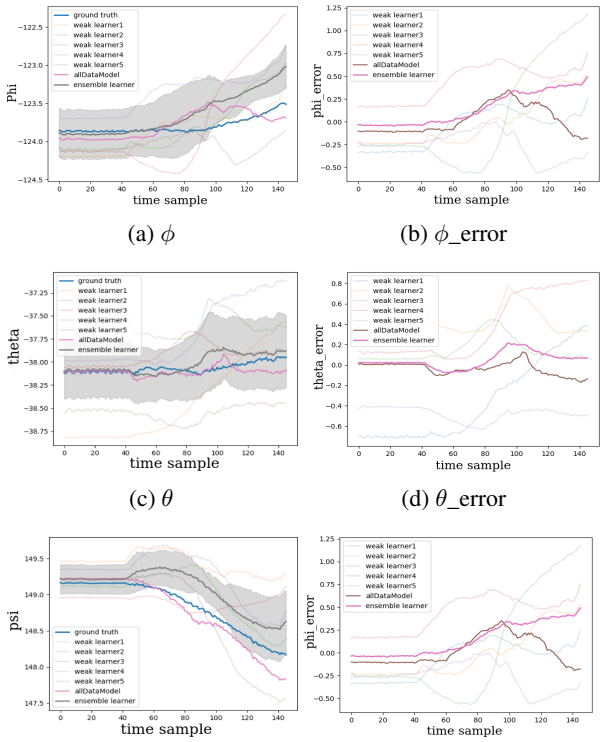
(c) Y (d) Y_error

(e) Z (f) Z_error

Figure 2: Position output for a sample task trajectory for CNN weak learners (blurred) and ensemble model (left column) with its 95% confidence interval (gray band) and corresponding tracking error values (right column).

Bagging for RNNs

As Table 3 shows using RNNs significantly enhanced the accuracy of the predictions. This stands to the reason that recurrent networks consider the time history of the system and are suitable for deep time series forecasting. As such,

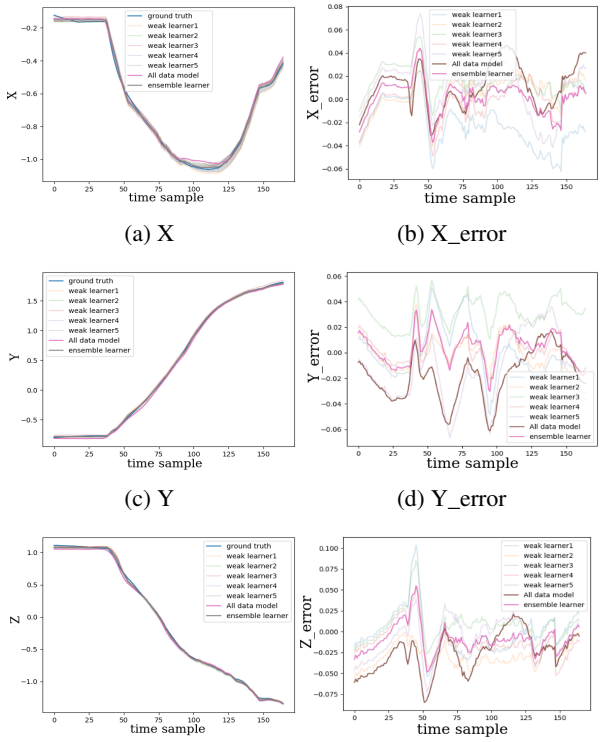


(a) ϕ (b) ϕ_error

(c) θ (d) θ_error

(e) ψ (f) ψ_error

Figure 3: Orientation output for a sample task trajectory for CNN weak learners (blurred) and ensemble model (left column) with its 95% confidence interval (gray band) and corresponding tracking error values (right column).



(a) X (b) X_error

(c) Y (d) Y_error

(e) Z (f) Z_error

Figure 4: RNN output for a sample task trajectory for weak learners (blurred) and ensemble model with its confidence interval (top row) and corresponding tracking error values (bottom row).

Figure 4 shows the position output for predictions and true values. It is clear that tracking accuracy is improved relative to feedforward models. Corresponding error values for tracking is also shown in the figure. Since all the position curves coincide, looking at error graphs gives better intuition about the performance of the ensemble model relative to the weak learner and the model which was trained on all of the dataset. In Figure 5 orientation output for ensemble RNN models is illustrated. A significant improvement can be observed relative to CNN models in Euler angles tracking performance. An interesting point is the confidence interval in this case is much more narrower than CNN outputs for orientation which denotes the lower uncertainty of the ensemble model.

KF-Recurrent Multi-Layer Perceptron

Kalman Filter iterates over observation and prediction. It includes state and covariance prediction and update stages at every step. Assume a discrete time system with F as state transition matrix, which takes state the x_k from step k to $k+1$ and H_k as measurement matrix (showing what components of state vector is available in the measurements), with the following state-space equation:

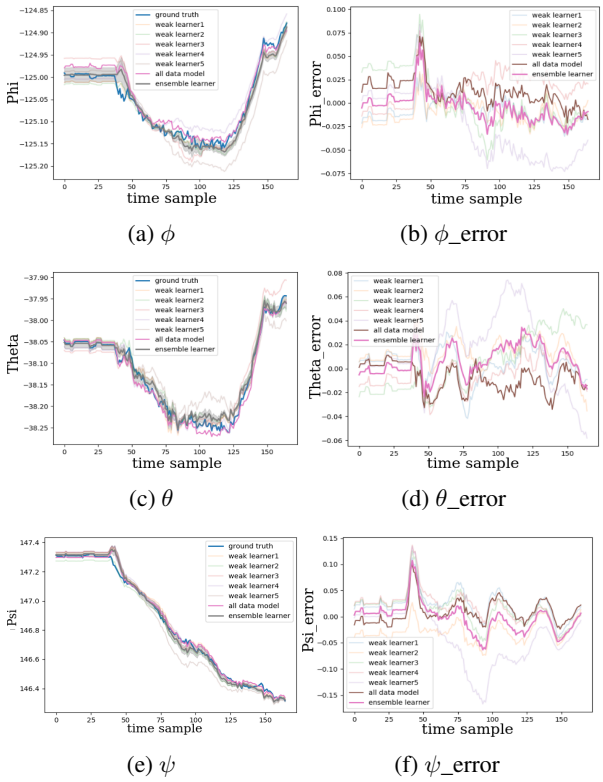
$$x_{k+1} = F_{k,k+1}x_k + w_k \tag{1}$$

$$y_k = H_k x_k + v_k \tag{2}$$

y_k is the observable at time k and w_k and v_k are independent, zero-mean, Gaussian noise processes of covariance matrices Q_k and R_k , respectively. Take $x_{k,k-1}$ as a priori estimation of state which is already available at step k by process model from equation (1) and $x_{k,k}$ as a posteriori state estimate which has to be calculated by the linear estimator. By having P_k as error covarince matrix, and K_k as Kalman gain matrix, Kalman filter loop is presented in Table 1.

Matrix I denotes the identity matrix. In each state, the output of the filter is resulted when the state and covariance are updated. By combining the process model and latest observation, the mean value of the distribution for each state variable is provided by the state matrix.

Figure 6 shows the block diagram for Kalman Filter training procedure for a recurrent multi layer perceptron network. For initialization the state variables $x_{0,0}$ can be chosen as reasonable values according to the first or first few observation(s). Arbitrary values can also be assigned as initial values; However, by assigning random values in some of the cases it takes the filter longer to converge until the tracking is stable. The covariance of the state can be initialised in a similar manner as described for state initialisation, which is either using the process noise covariance matrix Q or assigning a



(a) ϕ (b) ϕ_error

(c) θ (d) θ_error

(e) ψ (f) ψ_error

Figure 5: CNN output for a sample task trajectory for weak learners (blurred) and ensemble model with its confidence interval (top row) and corresponding tracking error values (bottom row).

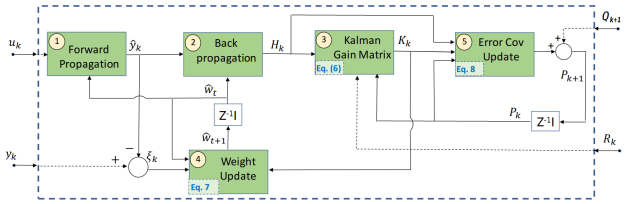


Figure 6: Signal flow diagram for KF neural network training.

constant matrix as its initial value (the identity matrix is more common); Which in this case it might take longer to become stable. The observation noise covariance R can be obtained via calibration.

The observing system can be managed to take a huge number of readings of a known ground truth state, and then the variance(s) can be obtained. Determination of process noise covariance Q can be more challenging. Some state transition equations may have a literal interpretation, which is useful to place bounds on the expected dynamic noise.

Kalman gains are calculated from the relative ratio of observation noise to the process noise. In practice, it is common to fix one of the noise covariance matrices and adjust the other to achieve the best performance. Assigning a constant value to Q and R, will result in K and P to converge to a constant value as well. This is because the observation has no influence on the gain and state covariance matrices. These terms assume that the relative noise distributions are given in Q and R. What they do is simply to combine those matrices to use in the state estimate.

In our CNN architecture, the output of the first dense layer with 80 units is considered as the input to our RMLP model, which is called latent vector in the paper. The architecture of our RMLP model includes dense layers and feed-backs from the output of each neuron that is connected to the input of the same neuron. The number of hidden layers and also the number of neurons in each layer can be considered as the hyper-parameters. In training process, we used one hidden layer with 20 neurons. We have applied (Extended) Kalman Filter algorithm to minimize our loss function and predict the next robot action.

Model	MAE	AE	Loss	E > 0.01
feed-forward	0.094	0.0136	0.0137	27.03
LSTM	0.031	0.0065	0.0054	7.18
GRU	0.024	0.0061	0.0059	9.67
RNN	0.021	0.0066	0.0123	15.34
CNN-Ensemble	0.080	0.0083	0.0109	22.71
GRU-Ensemble	0.019	0.0048	0.0049	5.12
KF-RMLP	**0.0014**	**0.0013**	**0.0018**	**3.43**

Table 3: Evaluation of KF-RMLP and comparison with LSTM, GRU and Ensemble models. The loss function is MSE and our approach outperforms in all of these mentioned terms.