Learning Motion Models for Robust Visual Object Tracking

Nathan Louis
Advisor: Jason J. Corso
Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI
natlouis@umich.edu

Abstract

Visual object tracking is an extensively researched area, but it remains an open problem due to many challenges resulting in tracking failures. The vast majority of state-of-the-art object trackers rely on deep neural networks and their many class-invariant features to find correspondences between consecutive frames. However, many of these trackers lack the ability to adapt in real-time to respond to temporal cues in videos. This failure to update the tracker's parameters results in losing the target as its movements and appearance varies. Recently there has also been work in estimating portions of the Kalman Filter framework using deep learning modules, as shown by Backprop-KF and LSTM-KF. We propose a deep learning tracking framework that uses state estimation theory to produce robust predictions in visual object tracking. We use a Siamese CNN to encode our observations and we follow LSTM-KF in using recurrent neural networks to produce a motion model and covariance estimates for the Kalman Filter update. We report our results on the ImageNet Video Object Detection dataset and compare to additional baselines.

1. Introduction

Visual object tracking is a classic problem with endless applications that can be employed in a variety of areas such as robotics, computer vision, the medical field, or video graphics. There has been extensive research with a wide-range of solutions aimed at solving these problems, from hand-crafted feature detectors to deep learning modules [3, 5, 12, 32, 35]. However there still exist significant challenges in visual object tracking. Some of the common failures include occlusion, change in appearance, illumination variance, and fast or blurred motion [30]. We also believe that an additional cause of failure is a lack of motion models in many modern trackers. For single target object tracking, we track an unknown object throughout a video sequence conditioned only on its initial position and appearance. In contrast, the multi-object tracking problem typically uses an object detector to find all known objects at each frame. The primary challenge becomes identity switching, re-identification, and missed detections. In our work we study single target object tracking, for the specific purpose of leveraging temporal cues in videos to improve tracking performance.

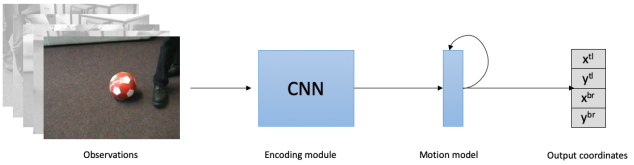

Figure 1: A high level overview of our complete tracking framework. The observations are video frames, the encoding module used is a Convolutional Neural Network (CNN), and we use a recurrent network architecture to perform our motion modeling. We output the target position for each video frame.

While temporal cues can be any information that exists across time in video, we focus on past positions of the object. We want to use the changes in position to model the object's motion, leveraging this to provide more accurate predictions of the target's location. To that end, we look towards work in state estimation theory, particularly Recursive Bayesian filters [31]. Recursive Bayesian filters are used to estimate a probability distribution given a state transition model, incoming measurements and observations. In our specific case, the learned motion model is the state transition model while incoming observations are used to produce estimated positions of the target. So in essence, we are filtering our position estimates to follow closely our learned motion model with the expectation of an improved localization accuracy. A high level overview for the concept of this framework is shown in Fig. 1.

Recent work by Coskun *et al.* [6] provides a deep net-

work architecture for learning the different modules in a Kalman Filter named the Long-Short Term Memory Kalman Filter (LSTM-KF). We follow a similar structure in our training regime with changes to the intermediate Kalman update calculations, the LSTM networks used, and the loss function formulation. Our model uses a Siamese network that regresses an object's target coordinates, whose output is filtered by an LSTM architecture that performs a Kalman update. In this work we propose a way to learn a motion model for a visual tracking framework, by first viewing the object's coordinate position as its state and using our model to predict the next state conditioned only on the current state and internal recurrent parameters. Our goal is to produce increasingly robust predictions using these state changes as temporal cues.

2. Related Works

Visual tracking has a dense history, evolving from low-level feature matching to high-level data-driven methods. Traditional methods primarily represent objects using points [2, 4], kernels [3, 5, 8, 13, 25, 32], or silhouettes [35]. Additionally, tracking schemes generally fall into generative or discriminative methods. Generative tracking techniques first defines the target appearance and then searches for candidate regions that best fit the appearance model [27]. Discriminative tracking learns to distinguish the target object from background regions. Discriminative approaches often include learning using both positive (target) and negative (non-target) regions of an image. They have empirically been shown to perform superior to generative methods and are utilized by many state-of-the-art tracking methods [1, 8, 13, 17, 27].

Classical trackers

The Kanade-Lucas-Tomasi (KLT) tracker [32] is a classic technique that performs tracking by solving the image registration problem. The KLT tracker assumes only a translation transformation between consecutive frames. Consequently, it does not handle occlusion or affine and homographic transformations. The Mean-shift Tracker [5] represents an ellipsoid region with a histogram of its RGB values, and it iteratively finds the next region with the most similarity. Bolme *et al.* [3] proposed an adaptive correlation filter-based tracker, MOSSE, that operates at real-time speeds and provides robustness to variations in lighting, scale, and pose. For each additional incoming frame, the filter is updated while down-weighting the effect of the previous frames. Kernelized Correlation Filters (KCF) [13] and many others [7, 26] build upon these results in correlation filter tracking. KCF particularly produces a kernelized version of the correlation filter by first viewing it as a ridge regression problem. Tracking-Learning-Detection (TLD) [17] is composed from a three-part tracking by detection method that additionally uses both positive and negative samples to continuously improve their detector. These classical trackers inherently learn online which is useful for adapting to novel videos, but they do not take advantage of the vast amount of offline video and image data. Hence, deep learning models have emerged to the forefront for tracking tasks primarily because of their ability to generalize to new data.

Deep Learning based trackers

MDNet [27] uses a small discriminative CNN to perform generic object tracking. While MDNet outperforms classical techniques (in terms of accuracy), its use of online fine-tuning limits the runtime to around 1 fps. GoTurn [12] is a Siamese network, based on the AlexNet architecture [21], that regresses to a set of bounding box coordinates for the target object. A Siamese network is an artificial neural network that operates on two inputs using the same set of shared weights. GoTurn does not perform well with certain occlusions and it does not learn from novel video sequences because the network weights are frozen. We use GoTurn as a baseline and its architecture for our Siamese model. Our aim is to include an LSTM-based motion model so our tracking framework can efficiently adapt to certain evolving target characteristics during evaluation. Other siamese networks [1, 23, 24, 33, 36] apply a cross-correlation layer after its convolutions instead of regressing directly to bounding box coordinates.

The majority of modern object trackers, rely purely on appearance to localize the target object in subsequent frames, and none provide motion models for the objects being tracked. There are a few works however, that have looked at capturing some temporal information through the use of Long Short Term Memory Networks (LSTMs) [10, 18] and Memory Networks [34]. However, in contrast to these methods, we propose to use the LSTMs to model different parts of a Kalman Filter to solve the tracking problem.

State estimation models

Traditionally, state estimation methods (Kalman Filtering, Extended Kalman Filtering, etc) are constructed with an explicit state transition and measurement model. There has been recent work in estimating different portions of the state estimation framework using deep learning. Deep Kalman Filtering [20] learns to model sequences of synthetic data in the presence of non-linear control inputs. Backprop KF [11] uses a CNN to process frame observations into states and estimates the measurement covariance. This was used for synthetic tracking experiments under linear Gaussian assumptions, which is not reasonable for highly non-linear

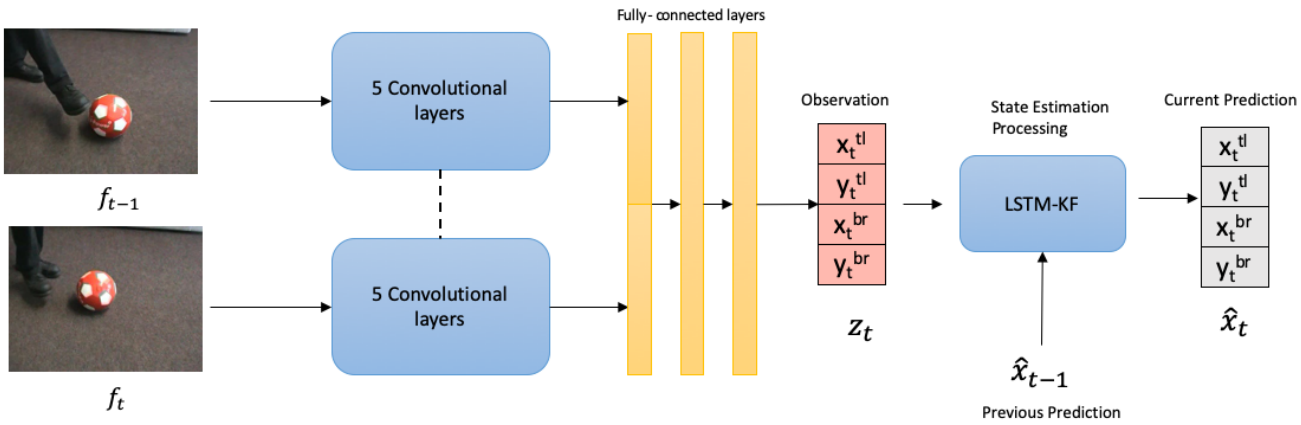

Figure 2: A detailed overview of our tracking framework. Two frames at f_{t-1} and f_t are passed through a shared set of convolutional layers, with f_{t-1} centered on last predicted position (or ground truth). Both outputs are concantenated and passed through a set of fully-connected layers to produce our observation $\mathbf{z}_t$, a measure for the target position on f_t. $\mathbf{z}_t$ along with $\hat{\mathbf{x}}_{t-1}$ (or $\mathbf{x}_{t-1}$) are input to the LSTM-KF unit to produce $\hat{\mathbf{x}}_t$, the corrected prediction on f_t.

real world video data. Objects in video sequences may undergo changes in acceleration, occlusion, out-of-plane rotations, etc. Both [20] and [11] operate under linear state transition assumptions and assume fixed state transition model or fixed state covariance. But Coskun *et al.* [6] produce LSTM-KF to attempt to learn the state transition models and covariances using LSTMs. In [6], they use LSTM modules to estimate the state transition model, state transition covariance, and measurement covariance for one-shot pose estimation in videos. The claim here is that the LSTMs will learn the relationship between states across time steps and produce a viable state model and covariance estimates across different time steps. In all of these cases, the estimates and models are input into the standard Kalman update equations. Our method is most similar to [6], where we aim to estimate several building blocks of the Kalman filter update equations purely from training data.

3. Background

A complete overview of State Estimation and Recurrent Neural Networks are included in Appendices A and B. We define our notation below.

Notation Overview

$\mathbf{x}_t \rightarrow$ Ground truth state	$\mathbf{P}_t \rightarrow$ State covariance
$\overline{\mathbf{x}}_t \rightarrow$ Belief prediction	$\overline{\mathbf{P}}_t \rightarrow$ Belief covariance
$\mathbf{z}_t \rightarrow$ Observation	$\mathbf{R}_t \rightarrow$ Observation covariance
$\hat{\mathbf{x}}_t \rightarrow$ Corrected prediction	$\hat{\mathbf{P}}_t \rightarrow$ Corrected covariance

Algorithm 1 Overview of method

input $V = [f_0, f_1, ..., f_T]$ {Input Video}
$X = [\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T]$ {Ground truth boxes}
initialize $P_0 \sim \mathcal{N}(0, I)$, $\hat{X} = \{\}$
for $t = 1$ **to** T **do**
  $\mathbf{z}_t = CNN(f_{t-1}, f_t)$
  $\overline{\mathbf{x}}_t = LSTM_f(\mathbf{x}_{t-1})$
  $Q_t = LSTM_Q(\overline{\mathbf{x}}_t)$
  $R_t = LSTM_R(\mathbf{z}_t)$
  $\hat{\mathbf{x}}_t, \hat{P}_t = kalman\_update(\overline{\mathbf{x}}_t, P_{t-1}, \mathbf{z}_t, Q_t, R_t)$
  $\hat{X}.append(\hat{\mathbf{x}}_t)$
  $P_t \leftarrow \hat{P}_t$
  if *curriculum_learning* **then**
    $\mathbf{x}_t \leftarrow \hat{\mathbf{x}}_t$ with $p = 0.5$
  end if
end for
return $\hat{X}$ {Final Predictions}

4. Method Overview

Our method takes inspiration from state estimation theory, filtering noisy outputs through a learned state transition model in order to provide robust behavior. An overview of our framework is shown in Algorithm 1. We have as input a video V of length T and its corresponding ground truth bounding boxes X. The output is $\hat{X}$, $T-1$ position predictions, but our framework can also produce the covariance of the state prediction, $\hat{\mathbf{P}}_t$ at each time step. As standard in single-target object tracking, $\hat{\mathbf{x}}_0 \in \mathbb{R}^4$ is initialized using the ground truth coordinate on the first frame, additionally we initialize $\hat{\mathbf{P}}_0 \in \mathbb{R}^{4\times4}$ as an identity matrix. Our 4-dimensional coordinate space for $\mathbf{x}_t$ is the top-left and

bottom-right coordinates of the bounding box.

We use a Siamese network, shown in Fig. 2, as our observation model. Like GoTurn [12], it regresses to a set of bounding box coordinates given a pair of consecutive frames as input. For each pair of input frames f_{t-1} and f_t we output an observation vector $\mathbf{z}_t \in \mathbb{R}^4$. In a trained tracking network $\mathbf{z}_t$ would be the final output prediction, but we treat this as our observation variable.

We use the LSTM-KF unit, shown in Fig 3, from [6] to perform many of the Kalman filter functions described in Appendix A. The inputs to the LSTM-KF are $\mathbf{z}_t$, $\hat{\mathbf{x}}_{t-1}$, and $\hat{\mathbf{P}}_{t-1}$ and the outputs are $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{P}}_t$. LSTM$_f$ is designed to perform as a state transition function. It produces the state belief prediction, $\overline{\mathbf{x}}_t$, at time t from the prior tracker prediction $\hat{\mathbf{x}}_{t-1}$. LSTM$_Q$ produces the state transition noise, Q_t, using $\overline{\mathbf{x}}_t$ as input. LSTM$_R$ produces the observation covariance, R_t, from $\mathbf{z}_t$. To provide covariance matrices that are positive-semidefinite, [6] simply produced diagonal matrices from LSTM$_Q$ and LSTM$_R$. But this assumes zero correlation between the elements of the state $\mathbf{x}_t$. This is a bad assumption, because in our problem the bounding boxes coordinates are highly-correlated and do not operate independently. The Cholesky decomposition states that any Hermitian positive-semi definite matrix can be decomposed into a matrix outer-product of a lower triangular matrix, $A = LL^T$, and the converse holds true for an invertible lower triangular matrix L [14]. So instead we first reshape our output into a lower triangular matrix and then multiply it with its transpose to produce the covariance matrices. In Appendix A, we see that because the state transition is modeled using a non-linear function we require the Jacobian to update our covariance estimates. [6] uses a linear layer from the output of LSTM$_Q$ to produce this Jacobian matrix G_t, but this is not guaranteed to be optimal or correct because it is not the true gradient of the state transition LSTM$_f$. Instead, we numerically calculate F_t as $\frac{\partial f}{\partial \hat{\mathbf{x}}_{t-1}}$ using the LSTM outputs at each iteration. Finally, all of these components are used as inputs into the standard Kalman update equations and produce $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{P}}_t$.

We experimented with both the L1 and L2 loss, but we found that models trained with the L1 loss provided a higher validation accuracy. We compute the loss between the ground truth $\mathbf{x}_t$ and the model prediction $\hat{\mathbf{x}}_t$:

$$\mathcal{L}(\theta) = \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{x}_t - \hat{\mathbf{x}}_t(\theta)\|_1 \quad (1)$$

Like [6] we also experimented with adding the error between the belief prediction, $\overline{\mathbf{x}}_t$, and the ground truth, $\mathbf{x}_t$, as a regularizer.

$$\mathcal{L}_{prediction}(\theta) = \lambda_1 \|\mathbf{x}_t - \overline{\mathbf{x}}_t(\theta)\|_1 \quad (2)$$

Through some hyper-parameter tuning, we empirically set $\lambda_1 = 1$. We do not have any ground truth data for the covariance matrix $\hat{\mathbf{P}}_t$, so we experimented with adding the Frobenius norm to encourage a lower covariance output. Our total loss now becomes:

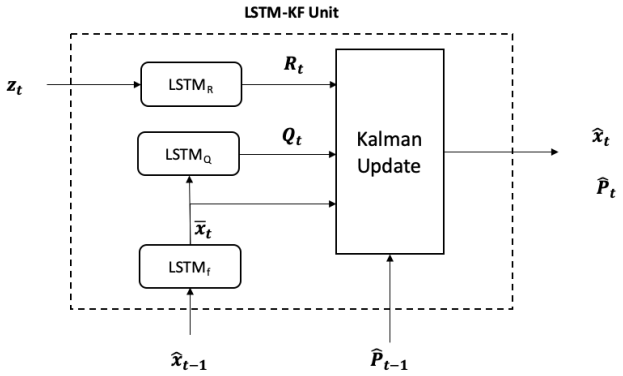

Figure 3: An inner block diagram of the LSTM-KF unit

$$\mathcal{L}_{total}(\theta) = \mathcal{L}(\theta) + \mathcal{L}_{prediction}(\theta) + \lambda_2 \|\hat{\mathbf{P}}_t\|_F \quad (3)$$

with $\lambda_2 = 0.01$.

4.1. Training Procedure

Unlike standard Siamese tracking methods, we train on video sequences rather than standalone image frame pairs. This means that we only update the model parameters after making predictions for the entire video sequence. During training, we sample video sequence lengths of 4, 8, and 16 frames. The loss in Equation 3 is computed after making a prediction for all frames $T-1$ frames in the sequence. We found that due to the high frame rate of the video sequences, many consecutive frames contain nearly identical content with very little movement. Instead of using all consecutive frames, for each video sequence we uniformly random sample an offset Δt between each frame, where $\Delta t \in [2, 32]$. For some experiments we produced our belief prediction $\overline{\mathbf{x}}_t$ from the ground truth $\mathbf{x}_{t-1}$. But we found this resulted in significant drift when evaluating the tracker on videos with longer sequences. The network was trained to output $\overline{\mathbf{x}}_t$ from ground truth data only, but not its own previous predictions which may contain some uncertainty. Instead we implement a variant of curriculum learning, similar to a method performed by [10]. During training, at each time step t we randomly select $\hat{\mathbf{x}}_{t-1}$ ($p = 0.5$), instead of $\mathbf{x}_{t-1}$, to produce our belief prediction $\overline{\mathbf{x}}_t$. While during validation, to closely model testing, we only use $\hat{\mathbf{x}}_{t-1}$ to compute the belief prediction. Table 2 shows the effects of using curriculum learning when evaluate on longer sequences, that what was previously trained. We also added gradient clipping to prevent the gradients from exploding when back-propagating through the LSTM units.

5. Experiments

5.1. Dataset(s)

Training and Validation

ILSVRC-VID The ImageNet Large Scale Visual Recognition Competition [29] provides an object detection for video dataset in addition to their image classification dataset. There are 30 object classes and all objects are annotated at every frame. From this dataset we use $26,400$ training video clips and $1,151$ validation video clips. Each clip tracks a single object, while several clips may be extracted from the same video.

5.2. Models

We trained with different models to compare against GoTurn [12] and LSTM-KF [6] as baselines. We denote LSTM-KF$_2$ and LSTM-KF$_3$ as our modifications to the LSTM-KF architecture. And finally we add a CNN+LSTM model for comparison. Similar to work done by [10], we will use a standard LSTM to recurrently update a hidden representation for the state. The details for each model are described below.

GoTurn$_1$ follows the same training regime as in [12]. Here the model trains on a single frame and an augmented version of that frame. Consequently, this trains on more data. The data augmentation is performed by applying a simulated motion and scale change to the original frame. This data augmentation, however, is a huge determining factor to the model's performance. The parameters for the motion and scale transformations were approximated from ALOV300++ [30], an existing visual tracking dataset, which may not generalize well to other videos. The model weights are updated for every pair of inputs, and we use the L1 loss between the model prediction and ground truth.

GoTurn$_2$ is trained with our training regime, and is directly comparable with the remaining models in terms of training data. Here we input pairs of consecutive frames, with some offset, from a video sequence into the model and only update the weights after producing predictions for the entire sequence. Again the loss is the L1 loss between the model predictions and the ground truths.

LSTM-KF$_1$ is implemented using the same details from the original paper [6]. The LSTMs used all have the same number of hidden units. Here we use Equation 3 as the loss function.

LSTM-KF$_2$ is our first modification to the LSTM-KF's formulations. Instead of a strictly diagonal covariance matrix output, we generate the covariance matrices using the Cholesky decomposition. We also compute the numerical Jacobian for LSTM$_f$ as F_t from the LSTM cell gates and outputs at each time step. In optimization problems the Jacobian matrix is often never computed directly because it may be too large to store into memory. But given our small state size, $\mathbf{x}_t \in \mathbb{R}^4$, we are able to calculate it without running into memory issues. Again the loss function is calculated from Equation 3.

LSTM-KF$_3$ is our second modification to LSTM-KF. In this variant, we make no attempt to approximate the Jacobian of LSTM$_f$. F_t is only used to update the belief covariance $\overline{\mathbf{P}}_t$, so instead we choose to approximate it using $\mathbf{P}_{t-1}$ and $\overline{\mathbf{x}}_t$ as input into a another non-linear function f_P.

$$\overline{\mathbf{P}}_t = f_P(\mathbf{P}_{t-1}, \overline{\mathbf{x}}_t) \tag{4}$$

We use an LSTM unit for f_P and the loss is also Equation 3.

CNN + LSTM This solution makes no assumption about the form of our state representation. Instead, we use the Siamese network to output a vector which serves as input directly to an LSTM. The output vector is the same dimension of the hidden states as the LSTMs. The LSTM outputs the bounding box coordinates, while updating the state representation using only its recurrent parameters. This model also uses the loss from Equation 3.

5.3. Implementation Details

All models were built and trained in PyTorch [28] and executed on TitanX or GeForce GTX 1080i GPUs. During training our convolutional layers are initialized with CaffeNet [16] weights pre-trained on the ImageNet image classification task [21]. Our fully-connected layers and LSTMs are randomly initialized using Xavier [9] initialization. The convolutional layers are frozen during training, while only the fully connected layers and LSTMs parameters are updated. We use a hidden state size of 128 for the LSTMs and an ADAM [19] optimizer with a learning rate of 1e-6. The model is evaluated on the validation set at the end of each training epoch, the model with the highest accuracy is saved. We trained all models for 50 epochs or until the validation accuracy saturates.

6. Results

Metric

We report our accuracy using mean Intersection-over-Union (mIoU). Given a predicted box A_p and a ground truth box A_g, $IoU = \frac{A_p \cap A_g}{A_p \cup A_g}$. This ratio tell us how well a predicted box is aligned with a ground truth box. mIoU is taken as the average of all given IOU measurements. For IoU, typically a value of 0.5 or greater is considered a success. Sample measures of IoU are shown in Fig 4.

[1]This model uses a different data loading strategy, hence it trains on more data.

Table 1: mIoU of the models on the validation set using the following sequence lengths.

Model	Train on	Length 4	Length 8	Length 16
GoTurn$_1$[1]	-	**0.8814**	**0.8821**	**0.8864**
GoTurn$_2$	Length 4	0.7213	0.7082	0.7041
	Length 8	0.6014	0.5818	0.5738
	Length 16	0.5963	0.5750	0.5651
LSTM-KF$_1$	Length 4	0.8645	0.8657	0.8702
	Length 8	0.8712	0.8703	0.8750
	Length 16	0.7869	0.8335	0.8578
LSTM-KF$_2$	Length 4	0.7658	0.7582	0.7571
	Length 8	0.7432	0.7326	0.7281
	Length 16	0.7814	0.7720	0.7701
LSTM-KF$_3$	Length 4	0.8075	0.6917	0.4365
	Length 8	0.8416	0.8451	0.8119
	Length 16	0.6948	0.7097	0.7361
CNN+LSTM	Length 4	0.8570	0.8574	0.8370
	Length 8	0.8675	0.8688	0.8727
	Length 16	**0.8813**	**0.8812**	**0.8842**

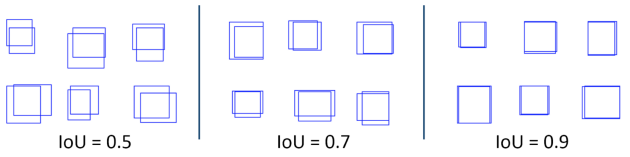

Figure 4: Visual samples of IoU boxes for the given values [37]

Quantitative

We calculate mIoU using all frames in the validation set and display the results in Table 1. All models, except GoTurn$_1$, were trained on video sequences with lengths $4, 8$, and 16 and evaluated at those lengths. For the models trained using our regime, CNN+LSTM outputs the highest mIoU for all sequence lengths. However LSTM-KF$_1$ performs higher on models trained on sequence lengths 4 and 8, while decreasing dramatically when trained on 16 length sequences. GoTurn$_1$ performs highest amongst all the models, but only slightly better than CNN+LSTM trained on 16 length sequences. It is important to note that GoTurn$_1$ used every frame in the training set, so it trains on approximately 4x as many video frames. It trains on 17x as many video frames when compared to models trained on 4 length video sequences.

In Table 2 we measure the effect of curriculum learning on the mIoU scores for longer sequences. The models in these experiments were only trained from 4 frame video sequences. While the mIoU scores from Table 1 are higher when evaluating on 4 frames, we see a significant improvement here when evaluating on 16 frames. Except for LSTM-KF$_1$, adding curriculum learning prevents a stark drop in accuracy.

Qualitative

We show qualitative results in Fig. 5 and Fig. 6. The bounding boxes shown represent the ground truth $\mathbf{x}_t$ (green), model output prediction $\hat{\mathbf{x}}_t$ (white), model belief prediction $\overline{\mathbf{x}}_t$ (red), and observation $\mathbf{z}_t$ (blue). When visualizing these outputs we notice that while the final output prediction may be correct, it seems that $\overline{\mathbf{x}}_t$ provides a very small contribution to the Kalman update. It does appear as informative to be used as a belief prediction because the area of the bounding box is small, and often times away from the target. Nonetheless the spatial structure of the bounding boxes provide some visual evidence of the Kalman update, where $\hat{\mathbf{x}}_t$ is somewhere between $\overline{\mathbf{x}}_t$ and $\mathbf{z}_t$.

7. Analysis

Representing our tracker state using its position coordinates allows us the advantage of using the Kalman update equations directly in an intuitive way. The problem

Table 2: Effect of curriculum learning. These were trained on 4 frames per video sequence using curriculum learning

Model	Length 4	Length 8	Length 16
LSTM-KF$_1$	0.7731	0.7265	0.6864
LSTM-KF$_2$	0.7785	0.7729	0.7744
LSTM-KF$_3$	0.7627	0.6846	0.6031
CNN + LSTM	**0.8529**	**0.8546**	**0.8579**

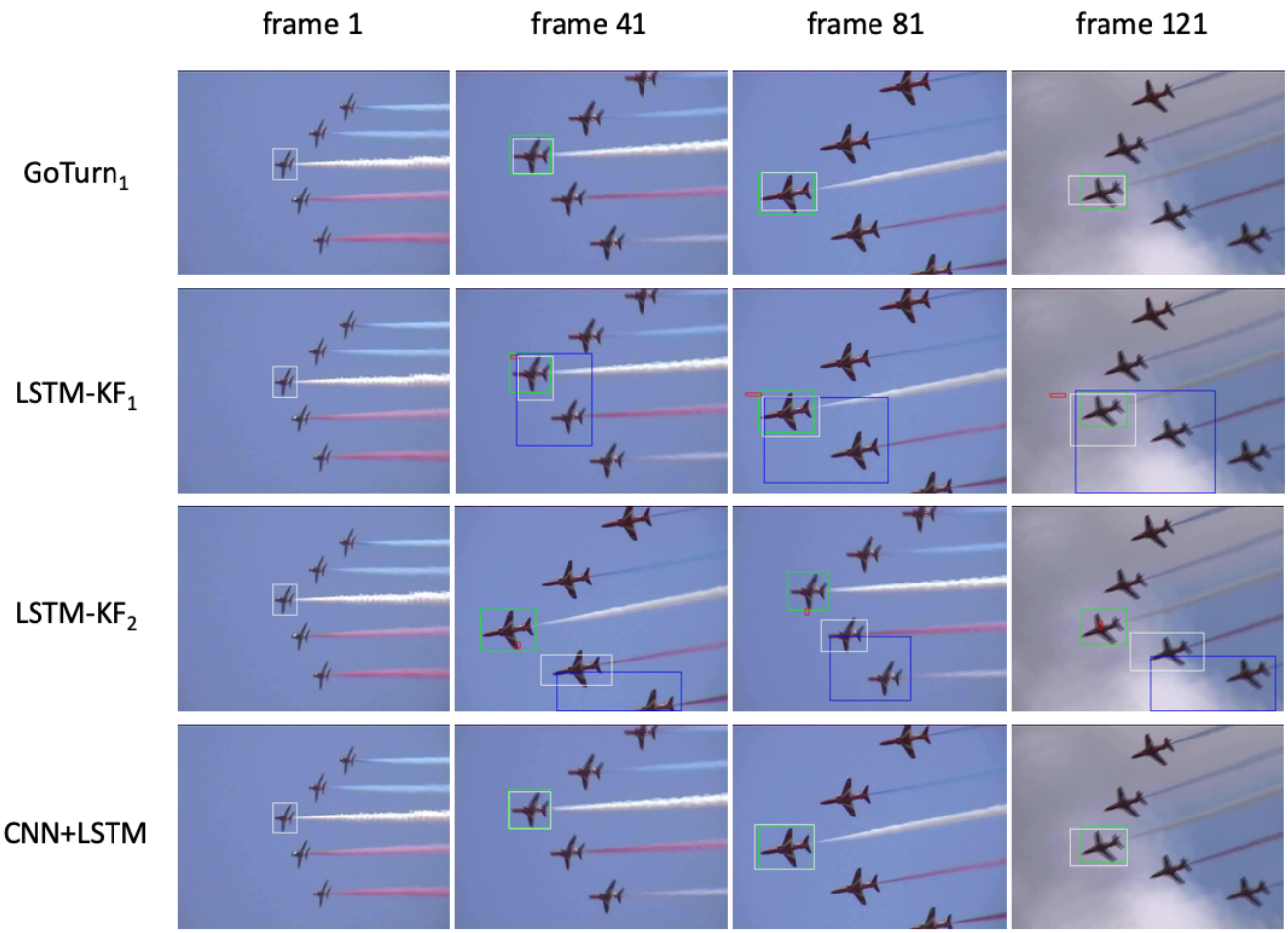

Figure 5: These are the visualized results from a video sequence from the validation set. While the intermediate values for the state estimation based trackers seem uninformative, the final tracking output is reasonable. The colors for the bounding boxes are $\mathbf{x}_t$ (green), $\hat{\mathbf{x}}_t$ (white), $\overline{\mathbf{x}}_t$ (red), and $\mathbf{z}_t$ (blue). (Best viewed in color)

reduces to something akin to a point tracker, and we can use the Kalman equations to update its position based on the learned motion dynamics. We use a deep network architecture and propose a way to learn these motion dynamics in a new tracking framework. From our quantitative results we've shown comparable performance compared to the GoTurn$_1$ and GoTurn$_2$ baselines, when training on magnitudes of less data. This comes as a direct benefit of using a recurrent structure to maintain a state representation. However, the CNN+LSTM model outperforms the other state estimatation based models that perform a Kalman update. The qualitative results do not match our expectations of the LSTM-KF tracking framework. From our experiments, the motion model doesn't appear to be learning an appropriate transition function between states, meanwhile the framework still manages to output the correct localization. This could be possibly driven by the contribution of $\mathbf{z}_t$ rather than $\overline{\mathbf{x}}_t$.

Traditional methods use position, velocity, and (sometimes) acceleration to represent the state. But we have no reliable way of measuring velocity or acceleration as the video frames may operate on different frame rates. Even so, using a 4-dimensional vector to represent our state has its fair share of disadvantages. The state is "information which summarizes the first observation in predicting the second observation" [22]. The qualitative results shows that this may not contain enough information to adequately predict the future state, even with the hidden layers of the LSTM. In problems like image classification, the feature representation for an image is typically on the order of 1024. Our 4-dimensional vector removes a lot of the visual spatial information from the CNN, in characterizing the pair of frames just using those coordinate vectors. But scaling up the dimensions of this state representation would prove very in-

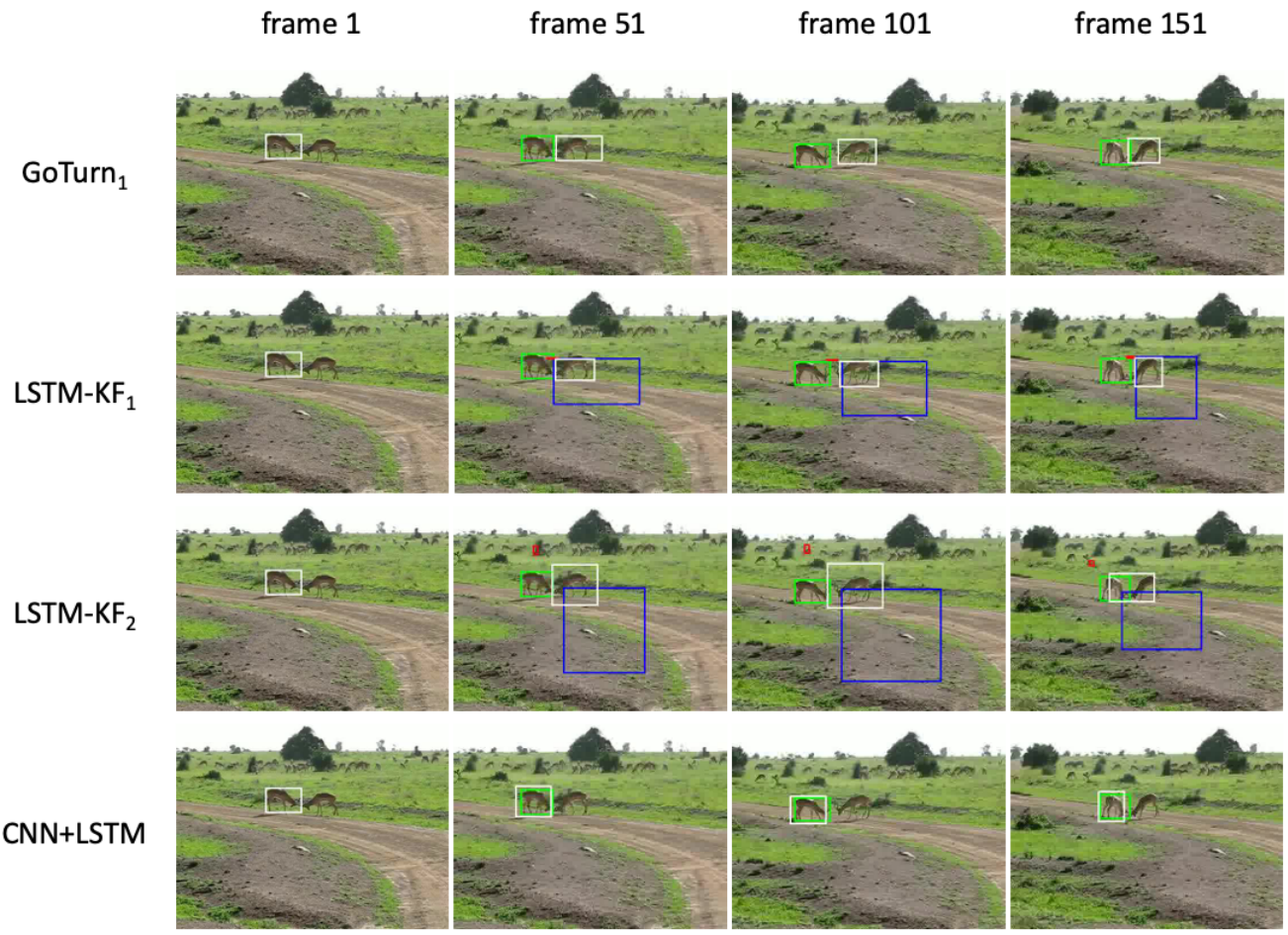

Figure 6: These are visualized results for a sequence from the validation set, all models except CNN+LSTM latch onto the wrong target during tracking. The colors for the bounding boxes are $\mathbf{x}_t$ (green), $\hat{\mathbf{x}}_t$ (white), $\overline{\mathbf{x}}_t$ (red), and $\mathbf{z}_t$ (blue). (Best viewed in color)

efficient given the matrix inverse in the Kalman update. On the other hand, the CNN+LSTM performs on-par or outperforms many of the preceding models without including matrix inverses or additional LSTM units. We can deduce that for the visual tracking task, updating the state representation using learned Kalman Filters may not bring significant gains given its additional cost.

8. Next Steps

We have empirically shown that leveraging temporal information during training provides a huge improvement in the performance of visual trackers. Further work can be done to learn how to abstract the state representation, in a way that retains more information from the CNN while finding ways to efficiently approximate a state estimation update. We can also find ways to train on longer sequences in a recurrent network architecture and restricting our preprocessing overhead by using unsupervised methods.

References

[1] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, *Fully-convolutional siamese networks for object tracking*, in European conference on computer vision, Springer, 2016, pp. 850–865. 2

[2] D. Beymer and K. Konolige, *Real-time tracking of multiple people using continuous detection*, in IEEE Frame Rate Workshop, Citeseer, 1999, pp. 1–8. 2

[3] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, *Visual object tracking using adaptive correlation filters*, in Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE, 2010, pp. 2544–2550. 1, 2

[4] T. J. Broida and R. Chellappa, *Estimation of object motion parameters from noisy images*, IEEE Transactions on Pattern Analysis & Machine Intelligence, (1986), pp. 90–99. 2

[5] D. Comaniciu, V. Ramesh, and P. Meer, *Kernel-based object tracking*, IEEE Transactions on pattern analysis and machine intelligence, 25 (2003), pp. 564–577. 1, 2

[6] H. Coskun, F. Achilles, R. S. DiPietro, N. Navab, and F. Tombari, *Long short-term memory kalman filters: Recurrent neural estimators for pose regularization.*, in ICCV, 2017, pp. 5525–5533. 1, 3, 4, 5

[7] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg, *Convolutional features for correlation filter based visual tracking*, in Proceedings of the IEEE International Conference on Computer Vision Workshops, 2015, pp. 58–66. 2

[8] ———, *Learning spatially regularized correlation filters for visual tracking*, in Proceedings of the IEEE international conference on computer vision, 2015, pp. 4310–4318. 2

[9] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in Proceedings of the thirteenth international conference on artificial intelligence and statistics, 2010, pp. 249–256. 5

[10] D. Gordon, A. Farhadi, and D. Fox, *Re 3: Re al-time recurrent regression networks for visual tracking of generic objects*, IEEE Robotics and Automation Letters, 3 (2018), pp. 788–795. 2, 4, 5

[11] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel, *Backprop kf: Learning discriminative deterministic state estimators*, in Advances in Neural Information Processing Systems, 2016, pp. 4376–4384. 2, 3

[12] D. Held, S. Thrun, and S. Savarese, *Learning to track at 100 fps with deep regression networks*, in European Conference on Computer Vision, Springer, 2016, pp. 749–765. 1, 2, 4, 5

[13] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, *High-speed tracking with kernelized correlation filters*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 37 (2015), pp. 583–596. 2

[14] N. J. Higham, *Cholesky factorization*, Wiley Interdisciplinary Reviews: Computational Statistics, 1 (2009), pp. 251–254. 4

[15] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural computation, 9 (1997), pp. 1735–1780. 11

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, *Caffe: Convolutional architecture for fast feature embedding*, in Proceedings of the 22nd ACM international conference on Multimedia, ACM, 2014, pp. 675–678. 5

[17] Z. Kalal, K. Mikolajczyk, J. Matas, et al., *Tracking-learning-detection*, IEEE transactions on pattern analysis and machine intelligence, 34 (2012), p. 1409. 2

[18] H.-I. Kim and R.-H. Park, *Residual lstm attention network for object tracking*, IEEE Signal Processing Letters, 25 (2018), pp. 1029–1033. 2

[19] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014). 5

[20] R. G. Krishnan, U. Shalit, and D. Sontag, *Deep kalman filters*, arXiv preprint arXiv:1511.05121, (2015). 2, 3

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097–1105. 2, 5

[22] J. Langford, R. Salakhutdinov, and T. Zhang, *Learning nonlinear dynamic models*, arXiv preprint arXiv:0905.3369, (2009). 7

[23] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, and J. Yan, *Siamrpn++: Evolution of siamese visual tracking with very deep networks*, arXiv preprint arXiv:1812.11703, (2018). 2

[24] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, *High performance visual tracking with siamese region proposal network*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8971–8980. 2

[25] A. Lukezic, T. Vojir, L. C. Zajc, J. Matas, and M. Kristan, *Discriminative correlation filter with channel and spatial reliability.*, in CVPR, vol. 6, 2017, p. 8. 2

[26] M. Mueller, N. Smith, and B. Ghanem, *Context-aware correlation filter tracking*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1396–1404. 2

[27] H. Nam and B. Han, *Learning multi-domain convolutional neural networks for visual tracking*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4293–4302. 2

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, *Automatic differentiation in pytorch*, (2017). 5

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision (IJCV), 115 (2015), pp. 211–252. 5

[30] A. W. SMEULDERS, D. M. CHU, R. CUCCHIARA, S. CALDERARA, A. DEHGHAN, AND M. SHAH, *Visual tracking: An experimental survey*, IEEE transactions on pattern analysis and machine intelligence, 36 (2014), pp. 1442–1468. 1, 5

[31] S. THRUN, W. BURGARD, AND D. FOX, *Probabilistic robotics*, MIT press, 2005. 1, 10

[32] C. TOMASI AND T. KANADE, *Detection and tracking of point features*, (1991). 1, 2

[33] J. VALMADRE, L. BERTINETTO, J. HENRIQUES, A. VEDALDI, AND P. H. TORR, *End-to-end representation learning for correlation filter based tracking*, in Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, IEEE, 2017, pp. 5000–5008. 2

[34] T. YANG AND A. B. CHAN, *Learning dynamic memory networks for object tracking*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 152–167. 2

[35] A. YILMAZ, O. JAVED, AND M. SHAH, *Object tracking: A survey*, Acm computing surveys (CSUR), 38 (2006), p. 13. 1, 2

[36] Z. ZHU, Q. WANG, B. LI, W. WU, J. YAN, AND W. HU, *Distractor-aware siamese networks for visual object tracking*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 101–117. 2

[37] C. L. ZITNICK AND P. DOLLÁR, *Edge boxes: Locating object proposals from edges*, in European conference on computer vision, Springer, 2014, pp. 391–405. 6

Appendix

A. State Estimation

Recursive state estimation algorithms are a popular tool which uses partial measurement data and probability methods to represent the state of a system that may not be directly observable. These algorithms provide a belief distribution over the possible world states [31]. The basis for many modern recursive state algorithms is the Bayes filter. For all Bayes filters the **belief** is represented as the posterior probability of a state variable conditioned on all available data: $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$ for a state variable x, measurement z, and control input u (if applicable). The belief is typically calculated from the **prediction**, computed before the measurement update at time t: $\overline{bel} = p(x_t|z_{1:t-1}, u_{1:t})$.

We will be using a popular implementation of the Bayes filter, the Kalman filter. The Kalman filter is part of the Gaussian filter family, which represents the beliefs of a system using a multivariate normal distribution. The state transition probability, $p(x_t|u_t, x_{t-1})$, used to produce the prediction from prior belief, is defined as:

$$\mathbf{x}_t = A_t\mathbf{x}_{t-1} + B_t\mathbf{u}_t + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, Q) \tag{5}$$

and similarly the measurement probability, $p(z_t|x_t)$, update is:

$$\mathbf{z}_t = H_t\mathbf{x}_t + \delta_t \qquad \delta_t \sim \mathcal{N}(0, R) \tag{6}$$

The Kalman filter operates under linear Gaussian assumptions, meaning the state transition probabilities and the measurement probabilities must be linear functions. A_t, B_t, and H_t are matrices used to transition from the current state onto the next. In following equations we remove $\mathbf{u}_t$ as we do not have a control input in our experiments. The variables ϵ_t and δ_t are added Gaussian noise, with Q and R as their respective covariances. Denoting our state as $\mathbf{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)$, the Kalman filter algorithm is shown as:

$$\overline{\boldsymbol{\mu}}_t = A_t\boldsymbol{\mu}_{t-1} \tag{7}$$
$$\overline{\Sigma}_t = A_t\Sigma_{t-1}A_t^\top + Q_t \tag{8}$$

where $\overline{\boldsymbol{\mu}}_t$ and $\overline{\Sigma}_t$ represent the belief predictions. The Kalman gain, K_t, along with the measurement update is defined as:

$$K_t = \overline{\Sigma}_t H_t^\top (H_t\overline{\Sigma}_t H_t^\top + R_t)^{-1} \tag{9}$$
$$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + K_t(\mathbf{z}_t - H_t\overline{\boldsymbol{\mu}}_t) \tag{10}$$
$$\Sigma_t = (I - K_tH_t)\overline{\Sigma}_t \tag{11}$$

Because of the linear Gaussian assumption, the standard Kalman filter cannot be used in many applications. However the Extended Kalman Filter (EKF) relaxes these assumptions allowing both the state transition and measurement probability to be non-linear, and linearizing them using a Taylor series expansion for the prediction update. Equations 7 and 8 now become:

$$\overline{\boldsymbol{\mu}}_t = g(\boldsymbol{\mu}_{t-1}) \tag{12}$$
$$\overline{\Sigma}_t = G_t\Sigma_{t-1}G_t^\top + Q_t \tag{13}$$

where g is a non-linear function and G_t is its first derivative *i.e* the Jacobian matrix. The remaining Kalman update equations follow similarly, with H_t now representing the Jacobian of the non-linear measurement update h. Equation 10 changes to:

$$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}} + K_t(\mathbf{z}_t - h(\overline{\boldsymbol{\mu}}_t)) \tag{14}$$

Using these non-linear functions, our implementation most resembles the EKF. We compute the measurement update from our CNN, a non-linear function approximator, Equation 6 can be appropriately written as $\mathbf{z}_t = h_\theta(\mathbf{x}_t)$. Additionally, we use an LSTM to perform our non-linear prediction update step.

B. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of artificial neural network that uses internal memory to process a sequence of inputs. This allows these networks to operate on sequential data: speech recognition, time series prediction, videos, etc. Traditional RNNs, however, suffer from vanishing or exploding gradients and have limited long-term dependencies. Long Short-Term Memory networks (LSTMs) [15] were introduced as a class of RNNs with a gated architecture that overcomes some of these problems. Instead of processing all sequence information, the network learns to control the flow of data between time steps. With the use of gates, the LSTM can learn to forget its current cell state and control how much data to input or output.

LSTM equations, $\odot$ represents element-wise multiplication:

$$\mathbf{i}_t = \sigma\left(W_x^i \mathbf{x}_t + W_h^i \mathbf{h}_{t-1} + \mathbf{b}^i\right) \tag{15}$$

$$\mathbf{f}_t = \sigma\left(W_x^f \mathbf{x}_t + W_h^f \mathbf{h}_{t-1} + \mathbf{b}^f\right) \tag{16}$$

$$\mathbf{o}_t = \sigma\left(W_x^o \mathbf{x}_t + W_h^o \mathbf{h}_{t-1} + \mathbf{b}^o\right) \tag{17}$$

$$\tilde{\mathbf{c}}_t = \tanh\left(W_x^c \mathbf{x}_t + W_h^c \mathbf{h}_{t-1} + \mathbf{b}^c\right) \tag{18}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{19}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right) \tag{20}$$

The input gate (15) controls the extent to which the input values are read into the LSTM cell, while the intermediate cell state (18) limits how much information is written into the cell. The forget gate (16) controls the values that remain from the previous cell state. $\mathbf{i}_t$, $\tilde{\mathbf{c}}_t$, and $\mathbf{f}_t$ are used along with the previous cell state $\mathbf{c}_{t-1}$ to produce the current cell state (19). The hidden state (20) is output from $\mathbf{c}_t$ and the output gate (17), which limits the values used from the non-linear activation. The last hidden state, $\mathbf{h}_t$, is typically used as the output of the LSTM.

Table 3: mIoU of the models on the validation set trained on 16 length sequence lengths.

	# frames trained on	Model	Length 4	Length 8	Length 16
Baselines	1.86×10^6	GoTurn$_1$	**0.8814**	**0.8821**	**0.8864**
	4.22×10^5	GoTurn$_2$	0.5963	0.5750	0.5651
	4.22×10^5	LSTM-KF$_1$	0.7869	0.8335	0.8578
Ours	4.22×10^5	LSTM-KF$_2$	0.7814	0.7720	0.7701
	4.22×10^5	LSTM-KF$_3$	0.6948	0.7097	0.7361
	4.22×10^5	CNN+LSTM	0.8813	0.8812	0.8842

Table 4: Effect of curriculum learning, models trained on 4 frames sequence lengths

	# frames trained on	Model	Length 4	Length 8	Length 16
Baseline	1.05×10^5	LSTM-KF$_1$	0.7731	0.7265	0.6864
Ours	1.05×10^5	LSTM-KF$_2$	0.7785	0.7729	0.7744
	1.05×10^5	LSTM-KF$_3$	0.7627	0.6846	0.6031
	1.05×10^5	CNN + LSTM	**0.8529**	**0.8546**	**0.8579**

Table 5: Without curriculum learning, models trained on 4 frames sequence lengths

	# frames trained on	Model	Length 4	Length 8	Length 16
Baseline	1.05×10^5	LSTM-KF$_1$	**0.8645**	**0.8657**	**0.8702**
Ours	1.05×10^5	LSTM-KF$_2$	0.7658	0.7582	0.7571
	1.05×10^5	LSTM-KF$_3$	0.8075	0.6917	0.4365
	1.05×10^5	CNN + LSTM	0.8570	0.8574	0.8370