**Proceedings of Proceedings of the ASME 2021
International Design Engineering Technical Conferences and &
Computers and Information in Engineering Conference
IDETC/CIE 2021
August 17-20, 2021, Virtual, Online**

DETC2021/MESA-68554

LSTM-ENABLED LEVEL CURVE TRACKING IN SCALAR FIELDS USING MULTIPLE MOBILE ROBOTS

Kunj J. Parikh
Computer Engineering Department
San Jose State University
San Jose, California, USA 95192
Email: kunjparikh6@gmail.com

Wencen Wu*
Computer Engineering Department
San Jose State University
San Jose, California, USA 95192
Email: wencen.wu@sjsu.edu

ABSTRACT

In this work, we investigate the problem of level curve tracking in unknown scalar fields using a limited number of mobile robots. We design and implement a long short term memory (LSTM) enabled control strategy for a mobile sensor network to detect and track desired level curves. Based on the existing work of cooperative Kalman filter, we design an LSTM-enhanced Kalman filter that utilizes the sensor measurements and a sequence of past fields and gradients to estimate the current field value and gradient. We also design an LSTM model to estimate the Hessian of the field. The LSTM enabled strategy has some benefits such as it can be trained offline on a collection of level curves in known fields prior to deployment, where the trained model will enable the mobile sensor network to track level curves in unknown fields for various applications. Another benefit is that we can train using larger resources to get more accurate models, while utilizing a limited number of resources when the mobile sensor network is deployed in production. Simulation results show that this LSTM enabled control strategy successfully tracks the level curve using a mobile multi-robot sensor network.

INTRODUCTION

Individual sensors or a mobile sensor network (MSN) can be deployed in an area to monitor the environment for a long period of time. In a mobile sensor network, the group of robots equipped with sensors is asked to simultaneously sense an unknown environment, and autonomously decide their trajectory. Some examples of autonomous MSNs include a swarm of unmanned aerial vehicles (UAVs) or unmanned underwater vehicles (UUVs), which are ideal candidates for exploring large-scale environments such as those encountered in the fields of ocean science and meteorology. The autonomous MSNs are involved in tasks including source seeking, level-curve tracking, mapping an unknown field, and many more. An example of level-curve tracking is tracking environmental boundaries [1, 2] as in monitoring the perimeter of a wildfire spreading over the land, tracking the perimeter of oil spills, understanding algae blooms, etc. Such a swarm can also sample a large-scale environment in order to chart an unknown field. There is considerable existing literature that solves some of the challenges involved in the level-curve tracking problem [3–7]. The robots carrying sensors have the capability to only take measurements at discrete locations and they have limited communication resources to share this information with each other. The authors in [4] designed a cooperative Kalman filter, which uses the instantaneous measurements of all the robots to cooperatively estimate the field gradient and the field hessian. They also design control strategies for the swarm trajectory and its shape which uses these estimated field parameters.

In addition to the instantaneous measurements (i.e. data about the current step), we can possibly use a sequence of histor-

*Address all correspondence to this author. The research work is supported by NSF grant CMMI-1917300.

ical data to infer the next step. Thus we apply machine learning technology to this problem in this work. Specifically, we apply Long Short Term Memory (LSTM) to study the collected state dynamics to produce state predictions. These state predictions are used with the sensor measurements to generate the state estimates, which are then used to control the trajectory and shape of the swarm (formation).

Using LSTM provides us with certain benefits compared to existing solutions that use the Kalman filter to estimate the field state. First, we don't need to explicitly derive the model or the state equations, which are possibly non-linear. If we follow the machine learning data-driven approach to collect, train, and simulate the system, without knowing a specific model, we can accomplish the same task. Second, using non-linear machine learning models, we can relax some assumptions made in existing approaches and thereby address more complex fields. Finally, using historical data and well-trained models can allow us to reduce the number of required robots to perform level-curve tracking when deploying the system in practice, while still enjoying the benefits of using a larger amount of robots in training. There is existing literature in which researchers have used LSTM-enhanced Kalman filter to tackle various challenges such as handling colored noise while taking measurements [8], tracking level-curve without using localization data [9], etc.

The contributions of this work are three-fold. First, we design and implement an LSTM enabled control strategy for a MSN to detect and track the desired level-curve. Here we use the estimates produced by an LSTM Kalman filter in the controllers to track the level-curve. Second, we develop a LSTM model to estimate the Hessian for the unknown curve, which is used as an input in the motion controller. Third, we conduct simulations to justify the approach and compare the results with existing approaches.

The rest of the paper is organized as follows. We first formulate the problem of level curve tracking in scalar fields. Then we introduce some necessary theoretical background for solving this problem. Next, we present the proposed LSTM-enabled level curve tracking method and the results and analysis. Finally, we make concluding remarks to discuss future work.

PROBLEM FORMULATION

In this section, we formulate the problem of boundary tracking using a MSN.

Scalar Field and Level Curve

We can model temperature in the ocean or in a wildfire using a scalar function of three dimensions. In order to simplify the problem, we assume this function is a static field as opposed to a time-varying dynamic field. Then, the temperature at each point in the ocean can be represented as a function over R^3, i.e. $z(r)$, where $r = (x,y,z)^T$ is a three-dimensional vector. In this work, we focus on 2-dimensional fields so we consider functions of the form $z(r)$, where $r = (x,y)^T \in R^2$. This function is not known *apriori*, but our problem is also not to fully estimate this field $\forall (x,y) \in R^2$. We assume the field is smooth and that the field can be approximated using second-order variations. Thus, the field characteristics at any location (x,y) can be defined using the field value $z(r)$, the field gradient $\nabla z(r)$, and the Hessian of the field $H(r)$ at that location. We assume that the field is smooth, its gradient is well-defined and bounded by a min and max value at all locations, i.e. $\nabla z_{min} < \nabla z(r) < \nabla z_{max}, \forall r \in R^2$.

A level set $L_c(z)$ of a function z with n variables is defined as a set of inputs for which the function takes a constant value c, i.e. $L_c(z) = \{(x_1,..,x_n) | z(x_1,..,x_n) = c\}$. For a two-dimensional field (i.e. $n = 2$) this level set is called a contour or a level-curve. Our problem is to find and track a level-curve given a desired constant value of the field. To again note, the field is not known to us, the only capability we have is to measure the field value at a given location. That is, we can measure z_1 at (x_1, y_1) using sensors. In addition, we deal with large-scale fields such as oceans, large wildfires, etc. so we can't sample all possible locations to measure the field.

Mobile Sensor Networks

With these constraints, we consider a scenario in which we deploy N robots in this field, and each robot has a sensor to measure the field at its own location, thus, form a MSN. We also call this collection of robots a swarm or a formation. We assume that the measurements are taken at discrete intervals, and we denote these local measurements by individual robots as z_i where $i \in \{1,..,N\}$. We further assume the robots can communicate with each other, so they can share their local field measurements with each other. In this work, we also assume that the robots know the relative locations of each other, which they can also communicate with others. We denote individual robot locations as r_i where $i \in \{1,..,N\}$ and $r_i \in R^2$, and we denote the center of the formation using r_c, where $r_c = \sum(r_i)/N$. Any or each of the robots can then use the collection of measured local field values and corresponding relative locations to estimate the characteristics of the overall field using methods that we will describe in subsequent sections.

Three Subproblems

For a MSN tasked with a level-curve tracking mission in an unknown scalar field, we are facing three subproblems. The first problem is to accurately estimate $z(r), \nabla z(r)$, and $H(r)$ using the collection of measurements z_i and corresponding locations r_i close to r. We take the center r_c as the point close to all the individual r_i to represent the formation. The second problem is to design a motion controller, which uses these estimates at the center $z_c, \nabla z_c$ and H_c, so that the formation center can detect

and track a level-curve with the desired field value, i.e. design a controller for the formation center so that $z_c = z_{desired}$. The third problem is how to control the motion of individual robots. In this paper, we will focus on the first subproblem by proposing an LSTM-enhanced Kalman Filter. We will use the existing motion controller and formation controller designed in [4] to achieve the motion and formation control.

THEORETICAL BACKGROUND

In this section, we introduce some theoretical background topics including sensor dynamics, motion controller design, and LSTM.

Level Curve Tracking Controller Design

Each robot is considered as a unit mass Newtonian particle that follows first-order dynamics. Thus, the formation center is modeled as $\dot{r}_c = u_c$, where u_c is the control applied to the formation center. We use the controller described in [4], which uses estimated z_c, ∇z_c, H_c of the center r_c and the desired level curve value $z_{desired}$ to control the trajectory of the formation. As illustrated in [4], representing the formation using Jacobi transformation allows the decoupling of the formation motion control and the formation shape control, thus, the controller design in this section only focuses on the control applied to the formation center with the assumption that the formation shape control is achieved.

As illustrated in Fig. 1, at any point along a level-curve, we can define an angle ϕ formed between the direction tangent to the field x_1 and the current heading of the formation center x, denoted by the green dot. Here y_1 is the same as the direction of the field gradient ∇z_c estimated earlier, and y is the direction orthogonal to the current heading x. (x_1, y_1) represents the Frenet-Serret frame of the level curve.

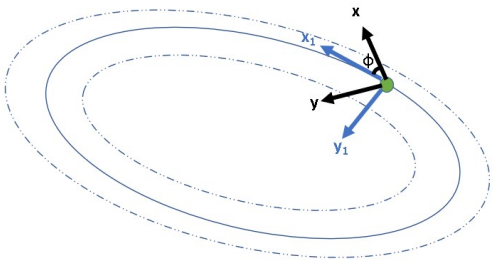

FIGURE 1: The coordinate frame of the formation center (x, y) and the Frenet-Serret frame of the level curve (x_1, y_1). ϕ is the angle between x and x_1.

Using the angle ϕ and assuming unit speed of the formation center, the steering control law can be described by

$$u_c = \kappa_1 \cos\phi + \kappa_2 \sin\phi - 2\tilde{f}(z_c)||\nabla z_c||\cos^2(\frac{\phi}{2}) + K_4 \sin(\frac{\phi}{2}), \tag{1}$$

where $\kappa_1 = -\frac{x_1^T \nabla^2 z_c x_1}{||\nabla z_c||}$, $\kappa_2 = \frac{x_1^T \nabla^2 z_c y_1}{||\nabla z_c||}$ in which $||\nabla z_c||$ is obtained using estimated field gradient ∇z_c. $\nabla^2 z_c$ is obtained using Hessian estimation. $\tilde{f}(z_c) = 0$ if the formation center r_c is on the level-curve, otherwise it models the external force which drives the formation towards the desired field value $z_{desired}$. K_4 is a constant gain parameter we can tune. The control law in (1) was derived using Lyapunov analysis with $\phi = 0$ as the equilibrium point.

Measurement Equation

To enable the motion controller, the field characteristics z_c, ∇z_c and H_c need to be estimated, given robot measurements z_i and r_i. In this paper, we use the measurement model used in [4], which is briefly summarized below.

First, the measurement p taken by each robot i at time k is modeled as:

$$p_{i,k} = z_{i,k} + w_{i,k} + n_{i,k}, \tag{2}$$

where $z_{i,k}$ is the value of the field at location $r_{i,k}$, $n_{i,k}$ is assumed to be i.i.d Gaussian noise, $w_{i,k}$ is spatially correlated noise. Assuming the field is smooth with respect to location r, we can approximate $z_{i,k}$ around formation center $r_{c,k}$ by its second-degree Taylor expansion.

$$z_{i,k} \approx z_{c,k} + (r_{i,k} - r_{c,k})^T \nabla z_{c,k} + \frac{1}{2}(r_{i,k} - r_{c,k})^T \nabla^2 z_{c,k}(z_{i,k} - r_{c,k}), \tag{3}$$

where $z_{c,k}$ is the field value at the formation center $r_{c,k}$, $\nabla z_{c,k}$ is the gradient of the field at $r_{c,k}$, and $\nabla^2 z_{c,k}$ is the second derivative of the field at $r_{c,k}$.

The state of the system can be defined as $\mathbf{s}_k = [z_{c,k}, \nabla_x z_{c,k}, \nabla_y z_{c,k}]$. The measurements $p_{i,k}$ of all the robots can be combined as a vector p_k. Similarly, combining noise elements for each robot's measurement, we get vectors w_k and n_k. The second derivative of the field $\nabla z_{c,k}$ can be estimated using hessian $\mathbf{H}_{c,k}$. The error in estimating the hessian is represented by e_k .

Using these notations and combining (1) and (2), we get,

$$p_k = C_k s_k + D_k \mathbf{H}_{c,k} + D_k e_k + w_k + n_k, \tag{4}$$

where $[C_k = \begin{bmatrix} 1 & (r_{1,k} - r_{c,k})^T \\ . & . \\ . & . \\ 1 & (r_{N,k} - r_{c,k})^T \end{bmatrix}$, and D_k is a $N \times 4$ matrix whose

i^{th} row is defined by $\frac{1}{2}((r_{i,k} - r_{c,k}) \otimes (r_{i,k} - r_{c,k}))^T$. As introduced in [4], with a state equation, a cooperative Kalman filter was designed to estimate the state $\mathbf{s}_k = [z_{c,k}, \nabla_x z_{c,k}, \nabla_y z_{c,k}]$ and a geometric approach was introduced to estimate the Hessian H_c based on the instantaneous measurements and the positions of the robots. The approach works well in unknown fields with a minimum of two robots (with rotating orientation) in 2D fields [4] and six robots in 3D fields [10]. We omit the derivation of the state equation due to the page limit.

Long Short Term Memory

A recurrent neural network (RNN) is a class of neural network where neurons form a directed graph along a temporal sequence. LSTM is a type of RNN that focuses on retaining long-term past information, learning by maintaining state and using the saved state with current input at any step. LSTMs do this by introducing various gates in the LSTM unit cell. Fig. 2 [11] shows the structure of one LSTM cell and how it is connected in a chain. The LSTM unit cell maintains the cell state C_t in addition to what the RNN unit cell does. The unit cell takes the inputs X_t, the previous cell state C_{t-1} and the previous output h_{t-1}. After performing some operations, it generates the current output h_t and the updated cell state C_t. In the figure, we can see the LSTM is composed of 4 gates in orange. Olah [12] provides a good overview of the LSTM unit cell structure.

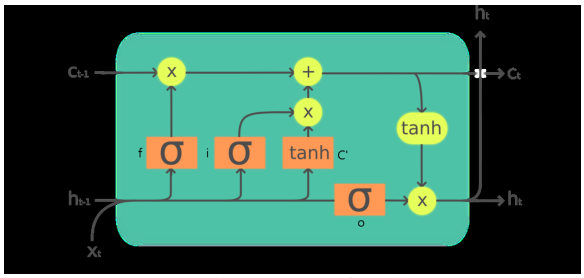

FIGURE 2: The LSTM unit cell structure.

The LSTM structure is represented mathematically by the following Equations (5). The first three equations represent the output of the forget gate f, the input gate i, and the cell update gate C' respectively. The fourth equation describes how the inputs of the first three gates are combined to update the cell state. This is represented by the $+$ operator in Fig. 2. The fifth equation is the output o_t of the output gate o,. And the last equation combines o_t and cell state C_t to compute the output of the cell h_t. The matrices W are the weights of each gate, and b is the bias term of each gate. The bias and the weights will be adjusted during LSTM training.

$$
\begin{aligned}
f_t &= \sigma(W_f.[h_{t-1}, X_t] + b_f), \\
i_t &= \sigma(W_i.[h_{t-1}, X_t] + b_i), \\
\tilde{C}_t &- \tanh(W_c.[h_{t-1}, X_t] + b_C), \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t, \\
o_t &= \sigma(W_o.[h_{t-1}, X_t] + b_o), \\
h_t &= o_t * \tanh(C_t).
\end{aligned}
\tag{5}
$$

LSTM and its variants like gated recurrent units (GRU, introduced by Cho et al. [13]) have seen huge success in the fields of speech recognition [14], text generation [15] , handwriting generation [16], and machine translation [17]. In the next section, we propose a solution that combines LSTM with the cooperative Kalman filter introduced in [4] to solve the overall problem of tracking a level-curve using a group of robots.

PROPOSED SOLUTION

As introduced previously, z_c, ∇z_c, and H_c at the center r_c need to be estimated at every time step in order to enable the motion control law (1). We propose that if we can first use historical data to train LSTM models offline, then in real deployment, we can extract the field information from the trained LSTM models directly. Thus, we can train using more resources, i.e., more robots, but deploy using fewer resources i.e., fewer robots, to measure the field at each step and consequently fewer robots to track the overall level-curve in resource-limited scenarios. For this purpose, we combine LSTM models with the cooperative Kalman filter. In this approach, LSTM models are used to predict the state, which helps with modeling complex fields and handling a different number of robot sensors, but we still rely on (or take benefit of) the Kalman filter to correct/tune the LSTM prediction.

We use the modified LSTM Kalman filter (LSTM-KF) [18], [19] described by the following Equations (6) - (10). The update steps, i.e., the first two equations, of the cooperative Kalman filter are replaced by two LSTMs. The LSTMs are given current and previous states and measurement errors as the input. In addition to the current data-point, we provide a number of previous data-points to the model controlled by the hyper-parameter called lag-window. The LSTMs estimate the state and measurement error through the first two Equations (6) and (7) giving us $s_{k(-)}$ and $P_{k(-)}$. The next three Equations (8) - (10) are left the same as in the cooperative Kalman filter, and they use the measurements taken by the robot sensors p_k and update the estimates to give us final state and measurement error $s_{k(+)}$ and $P_{k(+)}$.

$$
s_{k(-)} = LSTM_{State}(s_{k-1(+)}), \tag{6}
$$

$$P_{k(-)} = LSTM_{Error}(P_{k-1(+)}), \tag{7}$$

$$K_k = P_{k(-)}C_k^T[C_kP_{k(-)}C_k^T + D_kU_kD_k^T + R_k]^{-1}, \tag{8}$$

$$s_{k(+)} = s_{k(-)} + K_k(p_k - C_k s_{k(-)} - D_k H_{c,k}, \tag{9}$$

$$P_{k(+)}^{-1} = P_{k(-)}^{-1} + C_k^T[D_kU_kD_k^T + R_k]^{-1}C_k. \tag{10}$$

The field Hessian H_c is also needed in the controller for the formation center to track the level-curve. The approach described in [4] is an iterative numerical algorithm, which we need to repeat to obtain improved hessian estimates and it is derived for smooth fields. If we train another LSTM (which is inherently non-linear), we can get more accurate estimates of Hessian, and also expand Hessian estimation to more complex curves. So, we use a third LSTM to predict the Hessian at the current step. Similar to the other two LSTM models, we use the hyper-parameter lag-window to control how much of the history is presented to the LSTM to predict the current Hessian. Specifically, our 3rd LSTM's inputs are comprised of formation center r_c, field z_c, gradient ∇z_c, and previous hessian value at the center H_c^P, robot locations r_i, and measurements z_i. This is denoted by:

$$H_{c,k} = LSTM(r_c, z_c, \nabla z_c, r_i, z_i, H_{c,k-1}). \tag{11}$$

This predicted Hessian $H_{c,k}$ is used in the Kalman filter Equation (9) and formation center control law (1) described earlier. The flowchart of the complete proposed model is captured in Fig. 3.

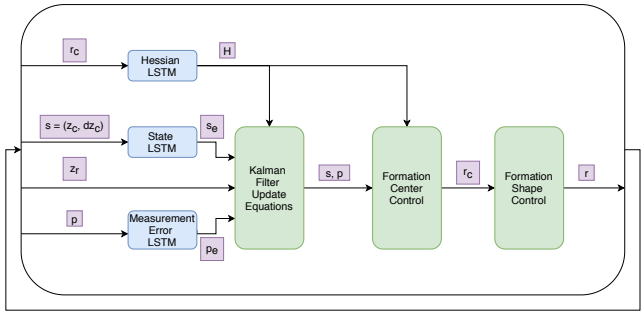

FIGURE 3: The proposed motion control flow using estimates generated through LSTM-KF.

Overview of the Proposed Algorithm

Data is needed for training the three LSTM models. For this reason, we first start with the existing cooperative Kalman filter and generate training data for smooth 2D fields. We deploy four robots in a symmetric formation to collect data in a number of training fields. One such training field is shown in Fig. 4. The figure shows an irregular field obtained using a 4^{th} order 2D polynomial. To collect data for training the LSTMs, given a starting point for the 4 robots, we set the desired field value to 144. At each point, the sensors measure field values. The previous state and measurements are passed to the cooperative Kalman filter to estimate the current state. The estimated state is passed to the formation center and formation shape controllers to update the trajectory to track desired the level-curve.

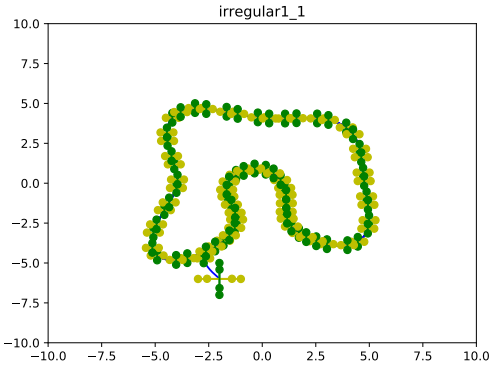

FIGURE 4: A irregular field tracked by 4 robots. The green dots represent robots r_1 and r_2, and yellow dots represent robots r_3 and r_4. The yellow and green lines are used to depict the formation of the robots r_i.

Once we have the training data set, we feed the current and a number of historical states $s_{k-i(+)}$ and errors $P_{k-i(+)}$ to the LSTM models as described in the first two Equations (6), (7) of the LSTM-KF model. The LSTM models predict the next state $s_{k(-)}$ and error $P_{k(-)}$. The actual next state and error collected as data earlier $(s_{k(+)}, P_{k(+)})$ are given as feedback to the LSTM to back-propagate the weights and learn the cooperative filter. The LSTMs are trained to minimize the root mean square error while predicting the state matrix and the error matrix.

Finally, we are ready to test the system with the trained LSTM models, i.e., use the LSTM-KF Equations (6), (7), (8), (9), and (10) to perform level curve tracking in unknown fields based on the flow shown in Fig. 3. Specifically, given the current and historical data, the LSTM models (Equations (6), (7)) predict the estimated current state $s_{k(-)}$ and the error matrix $P_{k(-)}$. The third LSTM model in Equation (11) provides the estimates of the Hessian. Then the robot measurements p_k update the state $s_{k(+)}$ and error matrix $P_{k(+)}$, i.e., correcting them with actual field

values using Equations (9), (10). These final values are provided to subsequent controllers to correct the formation-center's trajectory (Equation (1)) and to correct the shape of the formation for optimal data collection. Algorithm 1 describes the overall algorithm.

Algorithm 1 Motion Control using LSTM-KF

// Run cooperative Kalman Filter in training fields to estimate state and perform level curve tracking.
procedure COLLECT TRAINING DATA($s_{k-1}, P_{k-1}, z_i, H_k$)
// Use historical states, error covariances, current robot measurements, and estimated hessian to estimate the state.
procedure LSTM-KF($s_{k-j}, P_{k-j}, z_i, H_k$) ▷ Estimate the state
  $s_{k(-)} \Leftarrow$ Predict using s_{k-j} in Equation (6)
  $P_{k(-)} \Leftarrow$ Predict using P_{k-j} in Equation (7)
  $K_k \Leftarrow$ Calculate optimal gain using Equation (8)
  $s_{k(+)} \Leftarrow s_{k(-)}, p_k$. Update the prediction using Equation (9)
  $P_{k(+)} \Leftarrow P_{k(-)}$. Update the prediction using Equation (10)
  return $s_{k(+)}$
procedure MOTIONCONTROL ▷ Track Level Curve
  // Use lag-window as 50, i.e. 50 historical datapoints.
  Initialize lag-window lw
  Set time step T
  for $k = 1, .., T$ **do**
    $z_{k,i} \Leftarrow r_{k,i}$ Measure field at current sensor locations
    // Rearrange and present data for last lw time-steps to LSTMs
    $s_{k-j} \leftarrow [s_{k-lw}, s_{k-(lw-1)}, ...s_{k-1}]$
    $P_{k-j} \leftarrow [P_{k-lw}, P_{k-(lw-1)}, ..P_{k-1}]$
    // Estimate hessian.
    $H_k \Leftarrow$ Estimate hessian using Equation (11)
    $s_k \Leftarrow$ LSTM-KF($s_{k-j}, P_{k-j}, z_i, H_k$) ▷ Estimate State
    // Level curve tracking formation center controller
    $r_{k,c} \Leftarrow r_{(k-1,c)}, z_c, \nabla z_c, H_c$. ▷ Motion control Equation (1)
    // Level curve tracking formation center controller
    $r_{k,i} \Leftarrow r_{k,c}, r_{(k-1,i)}, z_c, \nabla z_c$ ▷ Formation shape control

EXPERIMENTAL RESULTS

In this section, we first discuss the implementation details of the algorithm, then illustrate the simulation results and analysis.

Data Collection

We first simulate tracking the level-curve with a MSN consisting of four robots using the cooperative Kalman filter to estimate the state and motion controllers to decide the trajectory. Throughout this simulation at each of the $T = 10000$ time-step k, we save the field values $z_{c,k}$, gradients $\nabla z_{c,k}$, error matrix P_k, Hessian H_k, sensor measurements $z_{i,k}$, robot locations $r_{i,k}$ and formation center $r_{c,k}$. We store this in a Python dataframe on disk. Next, we repeat this experiment for other field shapes. Thus, we end up with multiple dataframes - one for each shape. This series of dataframes gives us a sequence of data to train the proposed LSTM models. We consider a variety of shapes like circles of different polynomial degrees, ellipses with different coefficients, a few 4^{th} order 2D irregular polynomial shapes with varying coefficients. For example, one of the 4^{th} order polynomial shapes is:

$$(x^2 + 2y - 12)^2 + (K * x + y^2 - 17)^2 = 400,$$

where 400 is the desired field value $z_{desired}$. K is the coefficient we vary to generate more shapes.

We perform three main steps in data preparation for LSTMs. First, we flatten all the various vectors and matrices like state **s**, Hessian H_c, formation center r_c, robot locations r_i, sensor measurements z_r, error covariance matrix P into a single vector of 21 features for each time-step. Therefore, we get a 10000×21 matrix corresponding to the 10000 timesteps for a single level-curve shape. Second, we scale each feature using MinMaxScaler down to $[-1, +1]$ range. Finally, we reframe the data for supervised learning in the form of data X and labels Y. This reframing procedure is described next. For LSTMs, we need to decide on a hyper-parameter called lag-window. Lag-window is the amount of historical data we show to the LSTM to predict the current state. In the analysis section, we provide some insight into how we can get a range for this parameter in order to tune it. For example, if we choose lag-window to be 50, for the label at time-step k (y_k) we need to provide data from 50 previous time-steps ($[x_{k-50}, x_{k-49}, .., x_{k-1}]$). We re-organize the data in this fashion using a sliding window method. We end up with data X and labels Y matrices of the following format for training the state estimation LSTM:

$$X = \begin{bmatrix} [z_{c,0}, \nabla_x z_{c,0}, \nabla_y z_{c,0}] & .. & [z_{c,49}, \nabla_x z_{c,49}, \nabla_y z_{c,49}] \\ [z_{c,1}, \nabla_x z_{c,1}, \nabla_y z_{c,1}] & .. & [z_{c,50}, \nabla_x z_{c,50}, \nabla_y z_{c,50}] \\ . & . & . \\ . & . & . \\ & 9950 rows.. & \end{bmatrix},$$

$$Y = \begin{bmatrix} z_{c,50} & \nabla_x z_{c,50} & \nabla_y z_{c,50} \\ z_{c,51} & \nabla_x z_{c,51} & \nabla_y z_{c,51} \\ . & . & . \\ . & . & . \\ & 9950 rows.. & \end{bmatrix},$$

where X is of the shape $(9950 \times 50 \times 3)$ and Y is of the shape (9950×3). Similarly, for training the error co-variance LSTM,

in which each error co-variance is a (3×3) matrix, we obtain X and Y matrices for error co-variance of the shape $(9950 \times 50 \times 9)$ and (9950×9), respectively.

Training

After evaluating several LSTM structures, we use a simple model that consists of a single LSTM layer containing 40 neurons followed by a dense layer. We use the same configuration for all three LSTMs. The entire set of level-curves, each having X and Y matrices as described above, is split into the training, validation, and testing sets with the ratio 75:5:20. For the training stage, we use standard methods like early stopping, model checkpoints to save only the best model. We use model validation to improve the generalization of the trained models. We train for 200 epochs with early-stopping patience of 40 epochs, so if the validation-loss doesn't improve for 40 consecutive epochs we stop training further on that shape. Here, an epoch means going once over all the 9950 data points for a shape. This will be repeated 200 (epoch) times to improve the learned weights.

There are a few unique points we need to consider while training LSTMs. First, we can't shuffle the data because the pattern of the data matters. So, we train through each shape and use the learned model weights (instead of random weights) when starting with the next shape. Second, for each step during the training, we need to provide the data X in the (batch-size, lag-window, num-features) format and labels Y in (batch-size, num-features) format. Batch-size is selected to be 60 after manual grid-search to speed up the training but still derive a well-trained model. To account for the batch-size, the matrix X and Y described above need to be reorganized to be given 60 rows at a time.

Results and Analysis

While testing we try two important experiments. First, we test on unseen shapes using the same amount of robots during training and testing to evaluate the performance of the LSTM-KF algorithm. Second, we reduce the number of robots N in the testing stage. In this paper, we train the LSTM models using four robots, but we test using two robots. Note that the algorithm can be applied to $N >= 2$ robots in arbitrary formation as long as the correct Jacobi transform is defined. We use $N = 4$ and 2 for illustration purpose. Succeeding in the first experiment shows that we can handle unknown shapes, i.e. the trained LSTM model generalizes well. The second experiment shows that we can get away with using a less number of robots during testing, even though we used a larger number of robots during training to collect more accurate data. Thus, we can apply the algorithm in resource-limited situations.

The trajectories traced by a MSN with four robots using estimates produced by the traditional cooperative Kalman filter are shown in Fig. 5. In this figure, the violet line shows the desired level-curve, and the blue line shows the trajectory taken by the robots. The trajectories traced by a MSN with four robots using the field estimates predicted by the trained LSTM models are shown in Fig. 6. We observe that the robots don't trace the curve well for some of the concave curves and for these shapes the traditional cooperative Kalman filter performance is better than LSTM-KF in terms of providing better field and gradient estimates. However, the plots show a proof of concept that LSTM-KF can be used for estimating the field characteristics for level-curve tracking problems. The benefit of LSTM-KF is that it doesn't require explicitly deriving or modeling the state system, so for more complex scenarios such as dynamic time-varying fields the LSTM-KF method is more practical.

We plot in Fig. 7 (for the irregular1_8 shape in Fig. 6) the field and field gradients to show the error between actual shape and modeled trajectory. Here red plot shows the values we get if we follow the actual level-curve, which is obtained using the cooperative Kalman filter, green are the values we get using LSTM. The error (vertically) at each point (X-axis) determines the root mean square error. We try to reduce this root mean square error when we train the LSTM models.

In order to decide the lag-window hyper parameter, we plot the correlation across lagged values to judge for which values to train the LSTM on, i.e. how far into the history we need to go. For Fig. 8, the more diagonal the plot the higher the correlation. We see that till t-40 there's very high correlation, as we go back in history at t-160 the plot is pretty diffused, i.e. low correlation.

Similar information can be obtained from the auto-correlation plot for different lag values (Fig. 9). If the auto-correlation falls into the blue shaded cone it means the correlation can be a statistical fluke. If auto-correlation is outside the cone, there's a high chance that the correlation is not a statistical fluke. We see that until about 400 lags there is detectable auto-correlation for field gradients. The field value doesn't matter much because we are tracing a constant field value, but we see that up to about 200 lags field value has statistical auto-correlation. Note that in our dataset we also have the portion of the trajectory for which the robots have not reached the desired field value, so not all the field values are constant. We can see that in Fig. 7, we start with a high field value and approach the desired field in a limited number of steps.

Using the data from lag-correlation (Fig. 8) and auto-correlation (Fig. 9) plots, we decided to try 50 lag-steps as a good starting point to train LSTM on. That is we provide 50 historical state values to predict the single next state.

Finally, we show the results using two robot formation to track a level curve based on the LSTM models trained using four robots. Fig. 10 shows the trajectory taken by a two-robot formation around the circle. As we can observe there is some loss in accuracy especially towards the end of the circle, i.e. as we come closer to the starting point. The reason is that in the update/correct step of the LSTM-KF algorithm, measurements from only

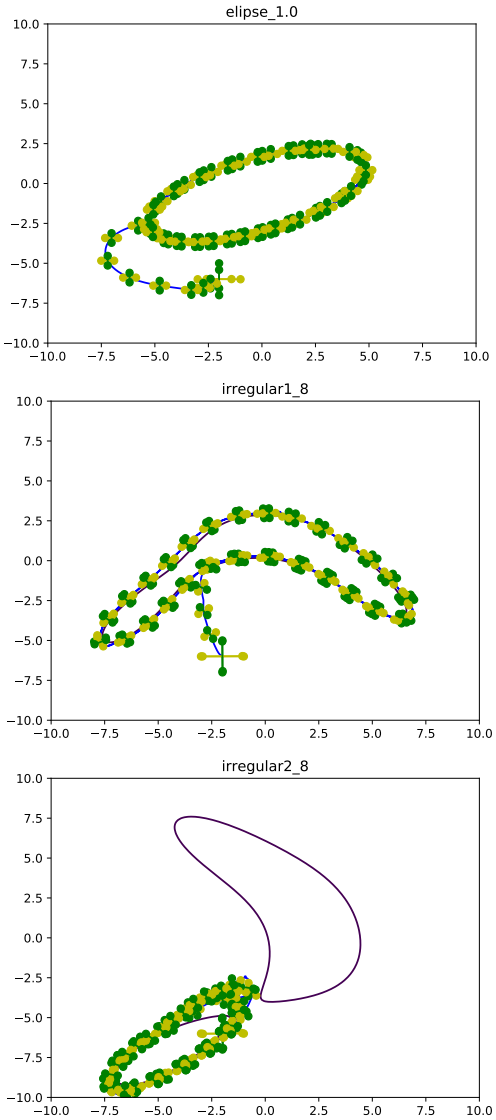

FIGURE 5: Fields traced by a group of four robots using cooperative Kalman Filter. The figures show 4 robots formation, green representing robots r_1 and r_2, yellow representing robots r_3 and r_4. The desired level-curve is shown in violet. The formation starts with the center r_c at (-2, -6). The blue level-curve is the actual trajectory followed by the simulated sensor network.

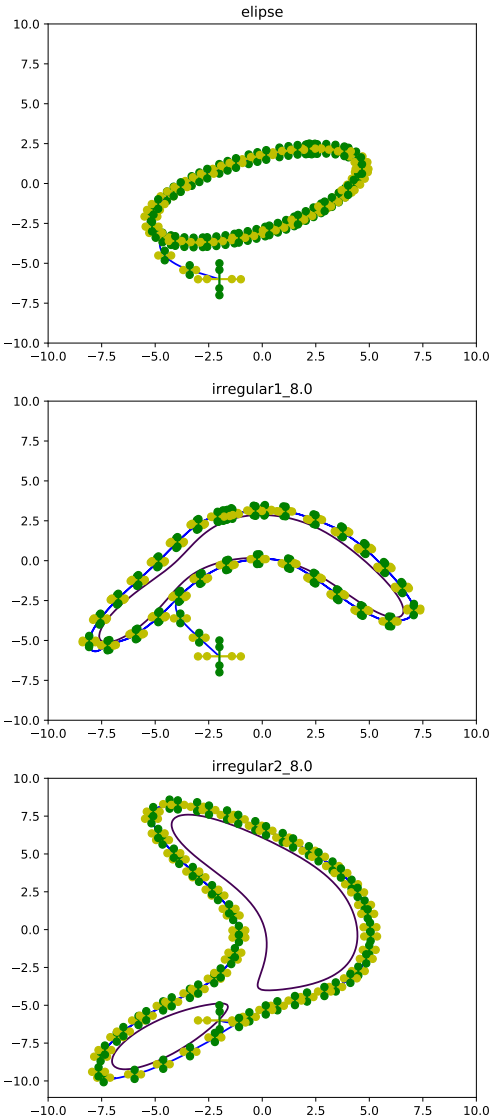

FIGURE 6: Fields traced by robots using the predictions by LSTM-KF.

2 robots will reduce the accuracy compared to using measurements from 4 robots in the training stage. Nevertheless, the result shows that we are able to deploy fewer robots in practice using the trained LSTM models.

CONCLUSIONS AND FUTURE WORK

In this work, we develop a LSTM-enabled cooperative level-curve tracking strategy, which allows a MSN to detect and track a level-curve in a scalar field with a desired level value. We show that using LSTM-enhanced Kalman filter allows us to use a sequence of states from history along with sensor measurements to estimate the current state, which allows users to train using more resources but deploy using fewer resources in resource-limited situations. In the future, we will improve the accuracy of the LSTM models so that the two robot formation tracks the level-curve more closely. In addition, we have considered static fields for this work, we will expand it to dynamic or time-varying fields.

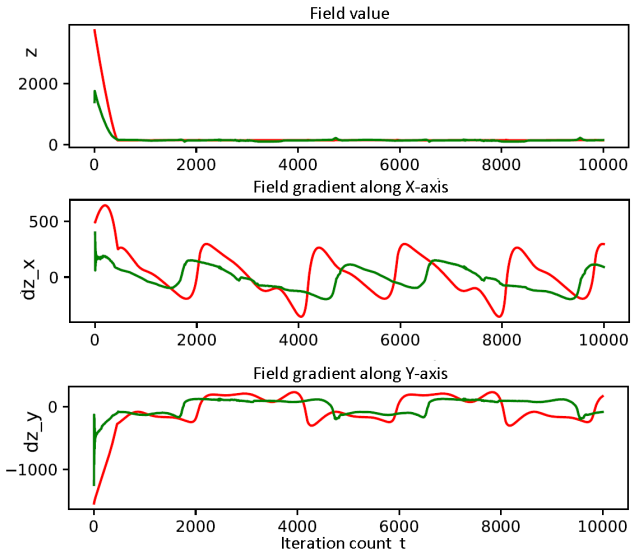

FIGURE 7: Field and field gradients along actual and predicted trajectories.

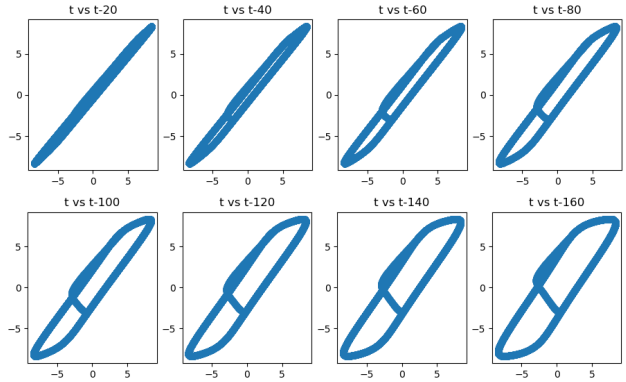

FIGURE 8: Correlation with different lags. Each of the 8 plots shows the correlation between the field gradient $\nabla_x z_c$ at the current time step vs $\nabla_x z_c$ at $t - t'$ time-step. A diagonal plot represents high correlation, while a diffused plot shows no correlation. As we increase the time-lag t' we observe vanishing correlation.

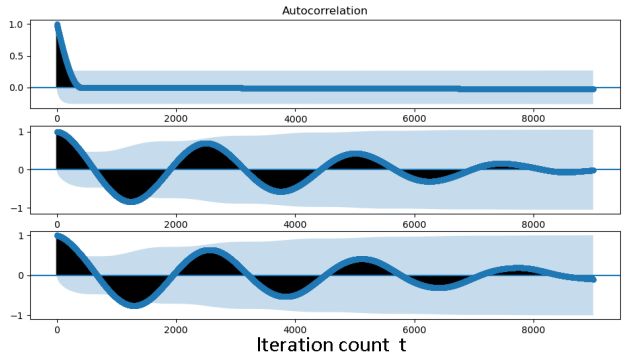

FIGURE 9: Auto correlation plot. From top to bottom, the plot shows auto-correlation for z_c (1^{st} plot), $\nabla_x z_c$ (2^{nd} plot), $\nabla_y z_c$ (3^{rd} plot) along the y-axis. The x-axis shows lag for auto-correlated sequence. The light blue cone represents the area within which the correlation is a statistical fluke. We observe that for lags $k <$ 200 the auto-correlation is outside the light blue cone and hence can be modelled.

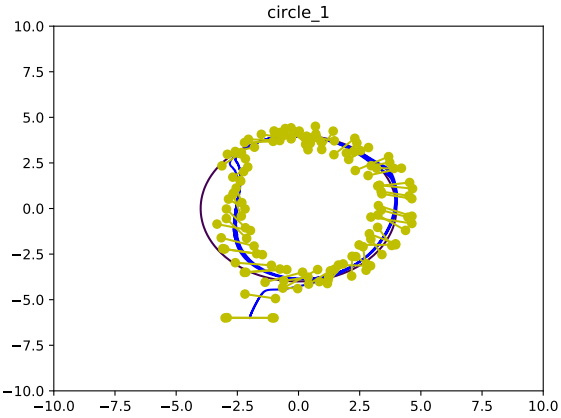

FIGURE 10: Trajectory traced by two robot formation around circular level-curve. Yellow dots represent the 2 robot sensors. An imaginary line connects them to the formation center r_c. Violet curve denotes the real level-curve. Blue shows the trajectory followed by the formation. Here the LSTM-KF model was trained using four robots and tested using two robots.

REFERENCES

[1] Marthaler, D., and Bertozzi, A. L., 2004. "Tracking environmental level sets with autonomous vehicles". In *Recent developments in cooperative control and optimization*. Springer, pp. 317–332.

[2] Leonard, N. E., Paley, D. A., Lekien, F., Sepulchre, R., Fratantoni, D. M., and Davis, R. E., 2007. "Collective motion, sensor networks, and ocean sampling". *Proceedings of the IEEE,* ***95***(1), pp. 48–74.

[3] Rosero García, E. E., 2017. "Cooperative source seeking and level curve tracking for multi-agent systems". PhD thesis, Technische Universität Hamburg-Harburg.

[4] Zhang, F., and Leonard, N. E., 2010. "Cooperative filters and control for cooperative exploration". *IEEE Transactions on Automatic Control,* ***55***(3), pp. 650–663.

[5] Williams, R. K., and Sukhatme, G. S., 2012. "Probabilistic spatial mapping and curve tracking in distributed multi-agent systems". In 2012 IEEE International Conference on Robotics and Automation, IEEE, pp. 1125–1130.

[6] Guruswamy, S., and Wu, W., 2020. "Cooperative level curve tracking in advection-diffusion fields". In 2020 5th International Conference on Automation, Control and Robotics Engineering (CACRE), IEEE, pp. 434–438.

[7] Chatterjee, S., and Wu, W., 2019. "A modular approach to level curve tracking with two nonholonomic mobile robots". In International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. 59292, American Society of Mechanical Engineers, p. V009T12A051.

[8] Choi, M., Sakthivel, R., and Chung, W. K., 2007. "Neural network-aided extended kalman filter for slam problem". In Proceedings 2007 IEEE International Conference on Robotics and Automation, IEEE, pp. 1686–1690.

[9] Zhang, Z., Al-Abri, S., Wu, W., and Zhang, F., 2020. "Level curve tracking without localization enabled by recurrent neural networks". In 2020 5th International Conference on Automation, Control and Robotics Engineering (CACRE), IEEE, pp. 759–763.

[10] Wu, W., and Zhang, F., 2011. "Cooperative exploration of level surfaces of three dimensional scalar fields". *Automatica,* ***47***(9), pp. 2044–2051.

[11] Chevalier, G., 2018. The lstm cell, May. Accessed on 10.31.2020.

[12] Olah, C. Understanding lstm networks. Accessed on 12.07.2019.

[13] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y., 2014. "Learning phrase representations using rnn encoder-decoder for statistical machine translation". *arXiv preprint arXiv:1406.1078*.

[14] Graves, A., and Jaitly, N., 2014. "Towards end-to-end speech recognition with recurrent neural networks". In International conference on machine learning, pp. 1764–1772.

[15] Sutskever, I., Martens, J., and Hinton, G. E., 2011. "Generating text with recurrent neural networks". In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 1017–1024.

[16] Graves, A., 2013. "Generating sequences with recurrent neural networks". *arXiv preprint arXiv:1308.0850*.

[17] Luong, M.-T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W., 2014. "Addressing the rare word problem in neural machine translation". *arXiv preprint arXiv:1410.8206*.

[18] Coskun, H., Achilles, F., DiPietro, R., Navab, N., and Tombari, F., 2017. "Long short-term memory kalman filters: Recurrent neural estimators for pose regularization". In Proceedings of the IEEE International Conference on Computer Vision, pp. 5524–5532.

[19] Zhang, Z., Hou, M., Zhang, F., and Edwards, C. R., 2019. "An LSTM based kalman filter for spatio-temporal ocean currents assimilation". In Proceedings of the International Conference on Underwater Networks & Systems, pp. 1–7.