

# Towards Unsupervised Learning, Classification and Prediction of Activities in a Stream-Based Framework

Mattias TIGER<sup>a</sup> and Fredrik HEINTZ<sup>a</sup>

<sup>a</sup>*Department of Computer and Information Science, Linköping University, Sweden*

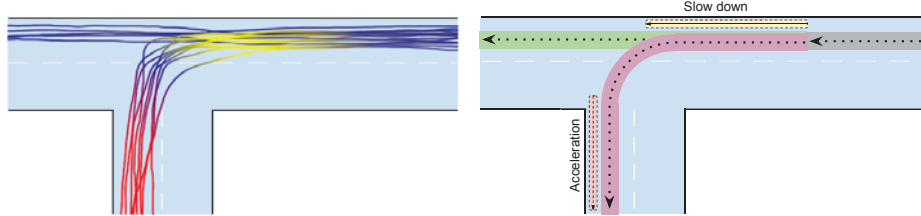
**Abstract.** Learning to recognize common activities such as traffic activities and robot behavior is an important and challenging problem related both to AI and robotics. We propose an unsupervised approach that takes streams of observations of objects as input and learns a probabilistic representation of the observed spatio-temporal activities and their causal relations. The dynamics of the activities are modeled using sparse Gaussian processes and their causal relations using a probabilistic graph. The learned model supports in limited form both estimating the most likely current activity and predicting the most likely future activities. The framework is evaluated by learning activities in a simulated traffic monitoring application and by learning the flight patterns of an autonomous quadcopter.

**Keywords.** Online Unsupervised Learning, Activity Recognition, Situation Awareness

## 1. Introduction

Learning and recognizing spatio-temporal activities from streams of observations is a central problem in artificial intelligence and robotics. Activity recognition is important for many applications including detecting human activities such as that a person has forgotten to take her medicine in an assisted home [1], monitoring telecommunication networks [2], and recognizing illegal or dangerous traffic behavior [3,4]. In each of the applications the input is a potentially large number of streams of observations coming from sensors and other sources. Based on these observations common patterns should be learned and later recognized, in real-time as new information becomes available.

We present the first step towards a stream-based framework for unsupervised learning, classification and prediction of spatio-temporal activities from incrementally available state trajectories. The framework currently supports unsupervised learning of atomic activities and their causal relations from observed state trajectories. Given a set of state space trajectories, the proposed framework segments the continuous trajectories into discrete activities and learns the statistical relations between activities as well as the continuous dynamics of each activity. The dynamics of the activities are modeled using sparse Gaussian Processes and their contextual relations using a causal probabilistic graph. The learned model is sparse and compact and supports in limited form both estimating the most likely current activity of an observed object and predicting the most likely future activities. This work extends our previous work [5] with improved activity updates that



**Figure 1.** An example of the input (left) and output (right) of the framework. The left figure shows observed trajectories. Blue is constant velocity, yellow is negative acceleration, and red is positive acceleration. The right figure shows three activities that the framework learned from these observations.

handles activities that are partially stationary as well as better support for merging. We also provide completely new experiments with real data from an autonomous quadcopter.

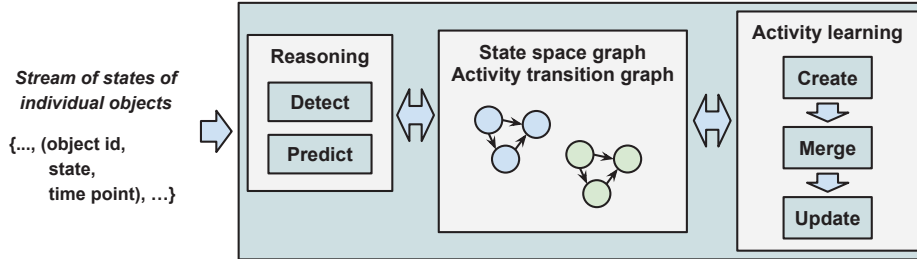
As an example, consider a traffic monitoring application where a UAV is given the task of monitoring traffic at a T-crossing. The UAV is equipped with a color and IR camera and is capable of detecting and tracking vehicles driving through the intersection [4]. The output of the vision-based tracking functionality consists of streams of states where a state represents the information about a single car at a single time-point and a stream represents the trajectory of a particular car over time (Figure 1, left). Based on this input the UAV should learn traffic activities such as driving straight or turning (Figure 1, right).

The framework is evaluated by learning activities in a simulated traffic monitoring application, as shown in Figure 1, and by learning the flight patterns of a quadcopter system. We also discuss how the framework can predict future activities.

The remainder of the paper is structured as follows. Related work in Section 2. Then we give an overview of the framework in Section 3. This is followed by three sections on modeling (Section 4), learning (Section 5) and detecting (Section 6) activities. Then we present our experimental results in Section 7 and conclude the paper in Section 8.

## 2. Related Work

There are many different approaches to activity recognition, both qualitative and quantitative. Here we focus on quantitative approaches since they are most related to our work. Among the trajectory based activity recognition approaches it is common to model the trajectories using splines with rectangular envelopes to model trajectories [6,7], chains of Gaussian distributions and Hidden Markov models [8] or using Gaussian processes [9]. Similar to our approach [7] divides trajectories into shorter segments and split trajectories at intersections, while [9] only considers trajectories beginning and ending out of sight and performs no segmentation of activities. Neither of the frameworks mentioned are suitable in a stream processing context with incrementally available information requiring online learning of new activities as they are observed. Our framework has the potential to continue to learn in the field on a robotic platform and adapt directly to new situations with regards to learning, classifying and predicting activities. Our framework supports not only detecting and keeping count of abnormal activities, but also learns those activities and collect statistics on their relations to other activities. This work extends to some extent that of [9], introducing temporal segmentation of trajectories into smaller



**Figure 2.** An overview of the framework. It has three modules *Reasoning*, *Activity learning* and the *Knowledge base* consisting of two graphs, which the other two modules make extensive use of. The *Activity learning* module takes observed trajectories and updates the graphs. It has three steps which are performed sequentially. The *Reasoning* module detects and predicts activities based on observations.

activity models similar to [7] to provide an online trajectory clustering methodology to activity learning and recognition.

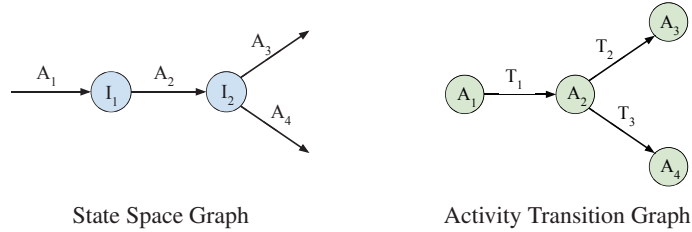
### 3. The Activity Learning Framework

The proposed framework (Figure 2) learns atomic activities and their relations based on streams of temporally ordered time-stamped probabilistic states for individual objects in an unsupervised manner. Using the learned activities, the framework classifies the current trajectory of an object to belong to the most likely chain of activities presently known by the framework. The framework can also predict how abnormal the current trajectory is and in a limited way how likely future activities are.

What we call an *atomic activity* is defined as a path through a state space (i.e. the 2D positions of a car) over time, from a start state at time  $t = 0.0$  to an end state at  $t = 1.0$ . It is represented by a state trajectory where some state variables are expected to change in a certain way while others may change freely. For example, an activity could be *slow down* where the velocity state variable is expected to decrease but where the angular velocity may vary freely. An activity model captures such continuous developments.

To learn a more abstract activity structure we capture the transitions between atomic activities in two graphs. The *Activity Transition Graph* captures the empirical transition probabilities observed and the *State Space Graph* captures the spatial connectivity of atomic activities by connecting them by *intersection points* which are state space transition points in which two or more activities are connected and separated in the state space. These graphs enable predicting future activities through probabilistic inference. An example is shown in Figure 3. The Activity Transition Graph is a causally directed discrete Markov chain. We currently only consider 1-order Markovian transitions. However, a natural extension is to use a higher order Markov chain which would allow activities to affect the likelihood of activities which are not directly before or after. This does however introduce some additional issues, such as how to keep the number of transitions sparse while finding out which should be kept. Such extensions are left for future work.

To handle activities on different scales the framework uses at the current stage two parameters,  $s_{threshold}$  and  $l_{threshold}$ . The parameter  $s_{threshold}$  is a threshold on the similarity between activities in the state space, based on the likelihood of one explaining the other. If an observed trajectory has a similarity below the threshold when compared to any other



**Figure 3.**  $A$  denotes an activity in both the State Space Graph and the Activity Transition Graph,  $I$  denotes an intersection point in the state space,  $T$  denotes a transition in the Activity Transition Graph. All transitions between  $A_2$ ,  $A_3$  and  $A_4$  can be seen as contained within the intersection point  $I_2$ .

activity, the observation is considered novel and unexplained. The parameter  $l_{threshold}$  is a threshold on the minimal length of an activity in the state space, which is used to limit the amount of detail in the learned model. In the work of [9] they detect abnormal trajectories by detecting the lack of a single dominant trajectory model explaining an observed trajectory. In our case a similar lack of dominance of activity models, or that several are almost equally likely, can be due to two things. Either due to abnormality (novelty for learning) or due to a transition between activities. By using the similarity threshold  $s_{threshold}$  we can distinguish between intervals that are either abnormal or transitional.

#### 4. Modeling Activities

An activity is here seen as a continuous trajectory in the state space, i.e. a mapping from time  $t$  to state  $y$ . A state can for example be a 2D position and a 2D velocity. Another example is the relative 3D position between a robot hand holding a coffee mug and the walls of a coffee machine. To get invariance in time,  $t$  is normalized to be between 0.0 and 1.0 and acts as a parametrization variable of  $y = f(t)$ . The trajectory is modeled as a Gaussian process model  $y = f_{GP}(t)$  and is used to classify new observations and to support prediction, similarly to [9]. The Gaussian process (GP) is a powerful modeling tool that can approximate arbitrary continuous non-linear functions, as well as provide a confidence measure of each function value, requiring only a finite number of data points. In our previous work we demonstrated an efficient way of modeling trajectories utilizing sparse Gaussian processes [10].

Given an observed state (e.g. the 2D position of a car), the parametrization variable  $t$  of the observed trajectory must be known in order to compare the observed state with a known activity to calculate how well the activity explains the observation. In [9] they assume this parametrization to be known by assuming that it is already normalized to the interval  $[0, 1]$  which can only occur after the whole trajectory has been observed. When performing stream reasoning the entire trajectory cannot be assumed to be present at once such that it can be normalized, because states are made available incrementally.

We estimate the parametrization for any observation to allow the system to operate with only incrementally available pieces of the observed trajectories. To make it possible to estimate  $t$  an inverse mapping is also modeled by a Gaussian Process model,  $t = g_{GP}(y) = f_{GP}^{-1}(y)$ , to estimate the mapping from state space to temporal parameter. For this mapping to be unique,  $f_{GP}$  must be bijective. This assumption restricts  $y = f_{GP}(t)$  such that it is not allowed to intersect itself at different time points.

This limitation can be circumvented by segmenting observed state trajectories into non-intersecting intervals and model each separately, something that is possible to do accurately and efficiently with the techniques in our previous work [10]. The inverse mapping is constructed by switching the input and the output,  $\mathbf{x}$  and  $\mathbf{y}$ , of  $f_{GP}$ . The purpose of the inverse mapping is to project a state space point  $z^* \sim \mathcal{N}(\mu_{z^*}, \sigma_{z^*}^2)$  onto the mean function of  $f_{GP}$ , thereby becoming the state space point  $y^* \sim \mathcal{N}(\mu_{y^*}, \sigma_{y^*}^2)$ . This is used when calculating the likelihood of a state on one trajectory being explained by a state on another. The calculation is performed by approximating  $z^*$  as  $\mu_{z^*}$  and  $t^*$  as  $\mu_{t^*}$  such that

$$y^* = f_{GP}(\mu_{t^*}) \quad \text{with} \quad t^* = g_{GP}(\mu_{z^*}). \quad (1)$$

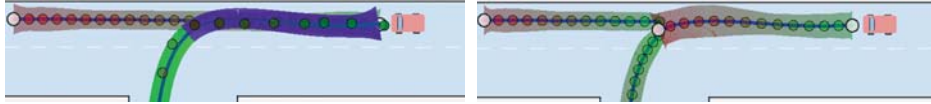
The former approximation can be improved by techniques for uncertain inputs to GPs [11]. The squared exponential kernel is used for both  $y = f_{GP}(t)$  and  $t = g_{GP}(y)$ , which restricts the applicable activities we try to model to ones that are smooth. Using the inverse mapping we can now define a point-wise similarity measure between two activities. Let  $A$  be an activity consisting of  $\{f_{GP}, g_{GP}\}$  and let  $z^* \sim \mathcal{N}(\mu_{z^*}, \sigma_{z^*}^2)$  be an observed state space point or a state space point on another activity. Assuming independent dimensions of the state space then the *local likelihood*[9] of  $x^*$  given  $A$  is

$$P(z^*|A) = P(z^*|y^*) = \prod_{n=0}^N P(z_n^*|y_n^*), \quad y^* = f_{GP}(g_{GP}(z^*)), \quad (2)$$

where  $N$  is the dimension of the state space and  $z^*$  and  $y^*$  are  $N$ -dimensional multivariate Gaussian distributions. The local likelihood is point-wise. The similarity between two sub-trajectories of two Gaussian processes is measured as the *global likelihood*[9] and it is calculated as the average local likelihood over the range of the two sub-trajectories.

## 5. Learning Activities

The framework learns connected atomic activity representations from observed trajectories by discriminating between intervals that are and are not similar enough to previously learned activities. Each activity differs spatially from all the other activities currently known. No two activities can have overlapping intervals in the state space, where the margin for overlap is controlled by the user parameter  $s_{threshold}$ . Example in Figure 4.

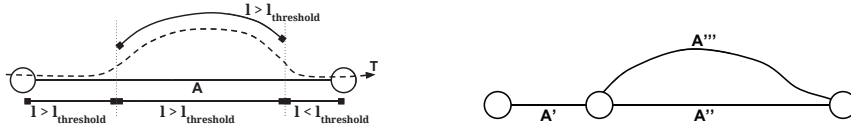


(a) An activity of driving straight and an observation of a left turn. Bright blue indicates interval for update. Bright green indicates interval which will result in creation of a new activity. (b) Now three activities after learning the left turn. The previous activity is now split at the point where the left turn was no longer explained sufficiently by the straight activity.

**Figure 4.** Example of learning a State Space Graph from 2D positions. Activity coloring is from green ( $t = 0.0$ ) to red ( $t = 1.0$ ). White circles are intersection points and colored circles are support points of  $f_{(GP)}$ .

The learning process updates the State Space Graph and the Activity Transition Graph based on an observed trajectory. The first step is to generate a GP model from the

observed trajectory, as explained in our previous work [10]. The next step is to classify the trajectory given the current set of known activity models. If a long enough interval of an observed trajectory lacks sufficient explanation given the known activities, then this interval is considered novel and is added as a new activity. Intervals that are explained sufficiently by a single activity more than by any other is used to update that activity with the new observed information. Sometimes long enough parts of activities become too similar, i.e. too close together in the state space, to be considered different activities. In those cases such parts of these activities are merged. This removes cases where intervals of the observation is sufficiently and on equal terms explained by more than one activity. The activity creation, updating and merging is illustrated in Figure 5 and Figure 6.



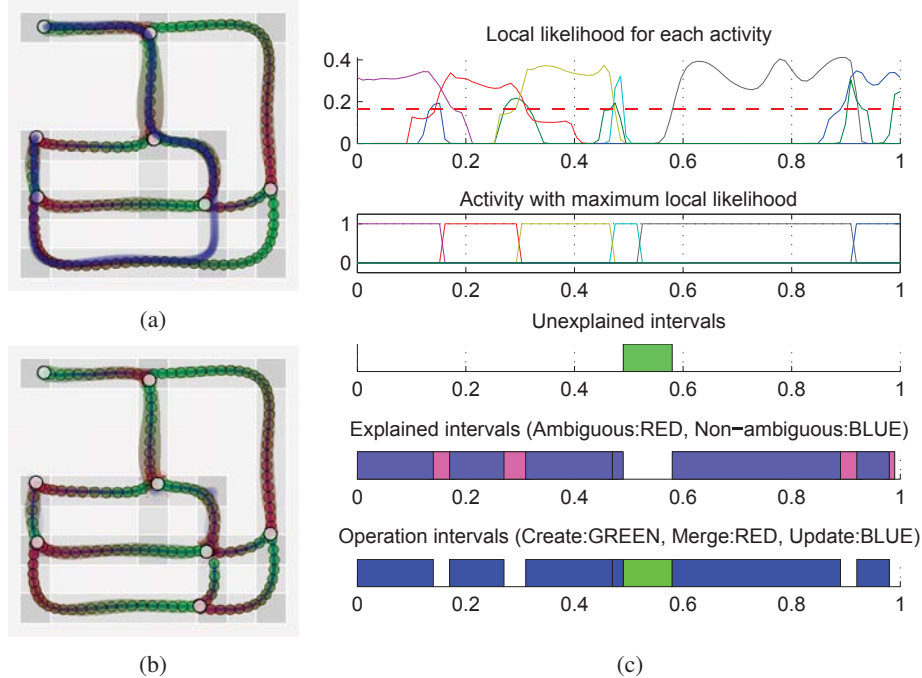
**Figure 5.** Illustrative example of the *create procedure* and *update procedure* for the observed trajectory  $T$  given the known activity  $A$  (left: before; right: after). The dotted lines indicate where the similarity of  $A$  and  $T$  is crossing the  $s_{threshold}$ .  $A''$  and  $A'''$  are created from  $A$  and  $T$  respectively.  $A'$  is created from  $A$  and  $T$  combined.



**Figure 6.** An illustrative example of the *merge procedure* of the two activities  $A_1$  and  $A_2$  when they have become too similar in the middle interval shown, resulting in five activities after the merge (left: before; right: after). The dotted lines indicate where the similarity of both  $A_2$  and  $A_1$  are exceeding the  $s_{threshold}$

Figure 7 illustrates the inner workings of the framework when updating its knowledge base with a new observed trajectory. Figure 7 (c) shows the different steps of the learning module. The top graph shows the local likelihood for each activity at each  $t$  of the observed trajectory. The red dashed line is the user parameter  $s_{threshold}$  and the y-axis of the plot is scaled to the power of 0.2. The second graph shows the activities with the maximum local likelihood. In the case where only one is above  $s_{threshold}$  it is a clear activity detection. If more than one activity is above  $s_{threshold}$  then that interval on  $t$  is ambiguous and it can be either a transition or multiple activities that have become too similar and should be merged. If no activity has a local likelihood above  $s_{threshold}$  for an interval then this interval is unexplained, meaning that a new, i.e. abnormal, activity is being observed. The third graph shows the intervals that are unexplained in the color green. The fourth graph shows the intervals that are found ambiguous (red) and the non-ambiguous (blue). The fifth and final graph shows the operations planned to be executed for each interval. After generating the set of planned operations making the knowledge base consistent, the create, merge and update operations are performed in that order.

Notice that at  $t = 0.5$  there is a transition between two activities (cyan and black) of best explaining the observation. Neither are however explaining it sufficiently, as no activity has a local likelihood that is above the limit  $s_{threshold}$ . The interval surrounding  $t = 0.5$  is therefore detected as a novel (abnormal) behavior rather than as a transition.



**Figure 7.** The State Space Graph (a) before and (b) after learning a new observed trajectory (shown as a thick blue line starting at the top-left of the graph). (c): The different stages of analysis performed by the framework.

There is a small transition right before  $t = 0.5$  where the observation crosses the cyan activity on the right side of the intersection point. This transition is however too small to be detected and both the yellow and the cyan activities are updated by the observation. The other occurrences of transitions are detected correctly and marked as ambiguous. Since these are transitions and not overlapping activities these shouldn't be merged.

## 6. Detecting and Predicting Activities

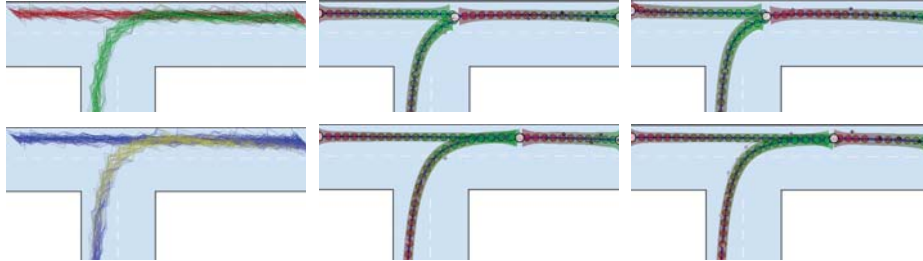
The State Space Graph and the Activity Transition Graph can be used in several ways to detect and predict activities. The statistical information of the evidence for each activity, similarity between any two activities and the probability of transition between activities is learned and available. Given the set of learned activities, it is straight-forward to calculate the activity chain from  $n$  steps in the past to  $m$  steps into the future that is most likely, least likely or anything in between. It is also possible to calculate the  $k$  most likely activity chains, with or without sharing activities at any step. Another possibility is to generate relevant events as part of the learning procedure. Events can for example be created when the learning procedure creates, updates and merges activities, or when an object has completed an activity due to the object crossing an intersection point.

## 7. Experiments

To verify that our approach works we have done two experiments. The first is a T-crossing scenario illustrating the benefits of using multiple types of data in the state, and the



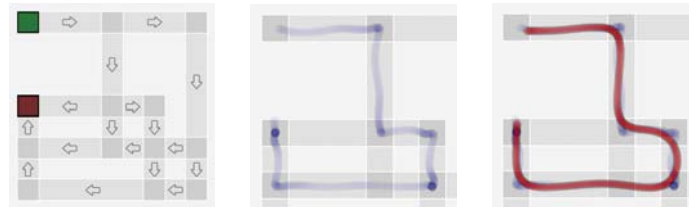
second is a UAV scenario demonstrating the framework’s capabilities to handle real data.



**Figure 8.** T-Crossing experiment. Top row: Positions only. Bottom row: Position and velocity. The figures in the middle show the three learned activities after two observed trajectories, while the figures to the right show the result after all observations. Blue indicate a higher speed than yellow.

The T-crossing scenario is similar to Figure 1 where two types of trajectories (50 each) are observed, straight trajectories and left turning trajectories. These are provided to the framework in a random order. The training data and the converged trajectories are shown in Figure 8. With position and velocity the turning activity starts earlier than with only position, allowing the framework to distinguish the two activities earlier.

The second experiment is a quadcopter UAV flying various patterns through a  $4 \times 4$  meter large grid with a top speed of  $0.5m/s$ . The UAV always flies in the same direction between each pair of points (Figure 9, Left). The branching in crossings are randomized and a total of 72 simulated flight trajectories and 25 real flight trajectories are used.

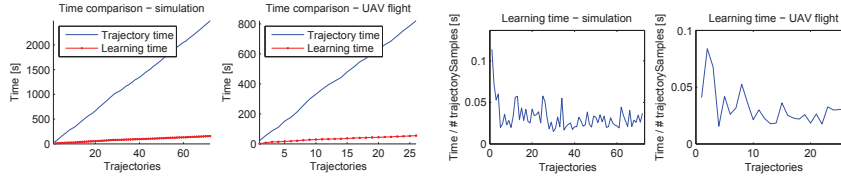


**Figure 9.** (Left) The graph of fly patterns used for the second experiment. A UAV always enters from the top-left green node and then arbitrarily traverses the graph until it reaches the red node. (Middle) An example trajectory with the positions of the UAV shown as blue circles. At some nodes the circles are stronger which indicate that the UAV has been standing still there. (Right) The red line is the mean of the GP model of the observed positions in the example when they have been down-sampled.

The input is full trajectories of 2D positions. All preprocessing and processing required is included in the presented results. Among this processing is the down-sampling of the dense positions in the observed trajectories. The framework in its current form requires that the state is continually changing. To handle the situation where for example the UAV is standing still, static data is removed by incrementally averaging all data points that are too close to each other resulting in a trajectory with evenly spaced data points in the state space. The minimum distance between two data points was set to  $15cm$ . This imposes a limitation on the resolution of detectable details. It does reduce the influence of the data points at the nodes have on the GP model and so the GP is a bit too round and sometimes misses the center of the nodes which can be seen to the right in Figure 9.

A comparison of the flight time of trajectories and the learning time of the trajectories is shown in Figure 10. The framework learns faster the more structure that has been





**Figure 10.** (First and second from left): A comparison between the cumulative flight time and time for learning trajectories. (First and second from right): The learning time of the framework for simulated and real flight after a number of trajectories. The x-axis indicates how many activities that has been learned so far.

learned. The experiment ran on a single 2.5 Ghz Gen.4 Intel i5 core. The learning rate is 11.89x and 10.39x real-time for the real and simulated trajectories respectively. This implies that the framework is feasible for real-time applications at least on small scales, or for learning activities from multiple simultaneously observed objects in real-time. The State Space Graph at various stages of the experiment is shown in Figure 11-12.

## 8. Conclusions and Future Work

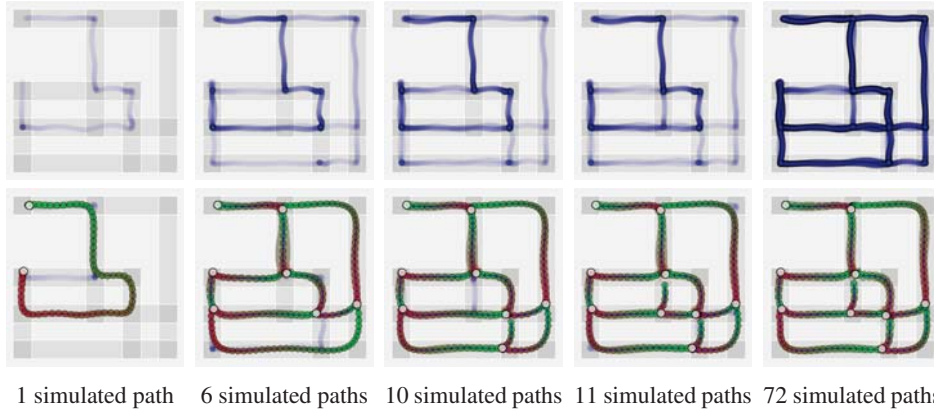
Learning and recognizing spatio-temporal activities from streams of observations is a central problem in artificial intelligence. In this paper, we have proposed an unsupervised stream processing framework that learns a probabilistic representation of observed spatio-temporal activities and their causal relations from observed trajectories of objects. An activity is represented by a Gaussian Process and the set of activities is represented by a State-Space Graph where the edges are activities and the nodes are intersection points between activities. To reason about chains of related activities an Activity Transition Graph is also learned which represents the statistical information about transitions between activities. To show that the framework works as intended, it has been applied to a small traffic monitoring example and to learning the flight patterns of a quadcopter UAV. We show that our framework is capable of learning the observed activities in real-time. The approach is linear both in the number of learned activities and in the dimension of the state space (assuming that the dimensions are independent). The GP model of each activity is cubic in the number of supporting points. However, by keeping a strict maximum number of supporting points, the complexity can be bounded. Due to the low complexity of the algorithms the solution is scalable to real-world situations.

Two aspects that we are interested in exploring further is how to learn complex activities as discrete entities and how to make the activities dependent on the context.

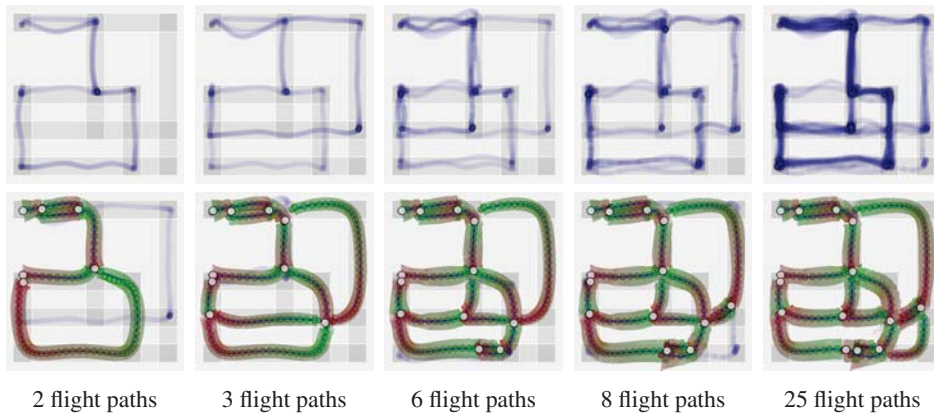
**Acknowledgments** This work is partially supported by grants from the National Graduate School in Computer Science, Sweden (CUGS), the Swedish Foundation for Strategic Research (SSF) project CUAS, the Swedish Research Council (VR) Linnaeus Center CADICS, ELLIIT Excellence Center at Linkoping-Lund for Information Technology, and the Center for Industrial Information Technology CENIIT.

## References

- [1] J. Aggarwal and M. Ryoo, "Human activity analysis: A review," *ACM Comp. Sur.*, vol. 43, no. 3, 2011.
- [2] C. Dousson and P. le Maigat, "Chronicle recognition improvement using temporal focusing and hierarchization," in *Proc. IJCAI*, 2007.



**Figure 11.** Learning of simulated flight trajectories. The top row shows the history of observed trajectories and the bottom row shows the incrementally learned model after various number of observed trajectories.



**Figure 12.** Learning of real flight trajectories. The top row shows the history of observed trajectories and the bottom row shows the incrementally learned model after various number of observed trajectories.

- [3] R. Gerber and H.-H. Nagel, "Representation of occurrences for road vehicle traffic," *AI*, vol. 172, 2008.
- [4] F. Heintz, P. Rudol, and P. Doherty, "From images to traffic behavior - a uav tracking and monitoring application," in *Proc. FUSION*, 2007.
- [5] M. Tiger and F. Heintz, "Towards learning and classifying spatio-temporal activities in a stream processing framework," in *Proc. of the Starting AI Researcher Symposium (STAIRS)*, 2014.
- [6] D. Makris and T. Ellis, "Learning semantic scene models from observing activity in visual surveillance." *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 35, no. 3, pp. 397–408, 2005.
- [7] N. Guillaume and X. Lerouvreur, "Unsupervised extraction of knowledge from S-AIS data for maritime situational awareness." in *Proc. FUSION*, 2013.
- [8] B. Morris and M. Trivedi, "Trajectory learning for activity understanding: Unsupervised, multilevel, and long-term adaptive approach," *IEEE Tran. PAMI*, vol. 33, no. 11, pp. 2287–2301, 2011.
- [9] K. Kim, D. Lee, and I. Essa, "Gaussian process regression flow for analysis of motion trajectories," in *Proc. ICCV*, 2011.
- [10] M. Tiger and F. Heintz, "Online sparse gaussian process regression for trajectory modeling," in *Proc. FUSION*, 2015.
- [11] A. Girard, C. Rasmussen, J. Candela, and R. Murray-Smith, "Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting." in *Proc. NIPS*, 2002.