Institutionen för datavetenskap
Department of Computer and Information Science

Examensarbete

Unsupervised Spatio-Temporal Activity Learning and Recognition in a Stream Processing Framework

Examensarbete utfört
vid Tekniska högskolan vid Linköpings universitet
av

Mattias Tiger

LIU-IDA/LITH-EX-A–14/059–SE

Linköping 2014-10-27

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings tekniska högskola
Linköpings universitet
581 83 Linköping

Unsupervised Spatio-Temporal Activity Learning and Recognition in a Stream Processing Framework

Examensarbete utfört
vid Tekniska högskolan vid Linköpings universitet
av

Mattias Tiger

LIU-IDA/LITH-EX-A–14/059–SE

Handledare: **Daniel de Leng**
IDA, Linköping University

Examinator: **Fredrik Heintz**
IDA, Linköping University

Linköping, 27 oktober 2014

Till min älskade mamma, som tålmodigt väntar på den dag hon kan få en tvättande, städande och diskande robot till sitt hem.

Avdelning, Institution
Division, Department

Artificial Intelligence and Integrated Computer Systems
Department of Computer and Information Science
SE-581 83 Linköping

Datum
Date

2014-10-27

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ _____

Rapporttyp
Report category

☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN
__

ISRN
LIU-IDA/LITH-EX-A–14/059–SE

Serietitel och serienummer
Title of series, numbering

ISSN
__

URL för elektronisk version

urn:nbn:se:liu:diva-111646

Titel
Title

Oövervakad maskininlärning och klassificering av spatio-temporala aktiviteter i ett strömbaserat ramverk

Unsupervised Spatio-Temporal Activity Learning and Recognition in a Stream Processing Framework

Författare
Author

Mattias Tiger

Sammanfattning
Abstract

Learning to recognize and predict common activities, performed by objects and observed by sensors, is an important and challenging problem related both to artificial intelligence and robotics. In this thesis, the general problem of dynamic adaptive situation awareness is considered and we argue for the need for an on-line bottom-up approach. A candidate for a bottom layer is proposed, which we consider to be capable of future extensions that can bring us closer towards the goal. We present a novel approach to adaptive activity learning, where a mapping between raw data and primitive activity concepts are learned and continuously improved on-line and unsupervised. The approach takes streams of observations of objects as input and learns a probabilistic representation of both 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 probabilistic graphs. The learned model supports both estimating the most likely activity and predicting the most likely future (and past) activities. Methods and ideas from a wide range of previous work are combined to provide a uniform and efficient way to handle a variety of common problems related to learning, classifying and predicting activities. The framework is evaluated both by learning activities in a simulated traffic monitoring application and by learning the flight patterns of an internally developed autonomous quadcopter system. The conclusion is that our framework is capable of learning the observed activities in real-time with good accuracy. We see this work as a step towards unsupervised learning of activities for robotic systems to adapt to new circumstances autonomously and to learn new activities on the fly that can be detected and predicted immediately.

Nyckelord
Keywords

Activity learning, Activity recognition, Activity prediction, Unsupervised On-line learning, Artificial Intelligence, Spatio-temporal, Stream processing, Sparse Gaussian process

Abstract

Learning to recognize and predict common activities, performed by objects and observed by sensors, is an important and challenging problem related both to artificial intelligence and robotics. In this thesis, the general problem of dynamic adaptive situation awareness is considered and we argue for the need for an on-line bottom-up approach. A candidate for a bottom layer is proposed, which we consider to be capable of future extensions that can bring us closer towards the goal. We present a novel approach to adaptive activity learning, where a mapping between raw data and primitive activity concepts are learned and continuously improved on-line and unsupervised. The approach takes streams of observations of objects as input and learns a probabilistic representation of both 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 probabilistic graphs. The learned model supports both estimating the most likely activity and predicting the most likely future (and past) activities. Methods and ideas from a wide range of previous work are combined to provide a uniform and efficient way to handle a variety of common problems related to learning, classifying and predicting activities. The framework is evaluated both by learning activities in a simulated traffic monitoring application and by learning the flight patterns of an internally developed autonomous quadcopter system. The conclusion is that our framework is capable of learning the observed activities in real-time with good accuracy. We see this work as a step towards unsupervised learning of activities for robotic systems to adapt to new circumstances autonomously and to learn new activities on the fly that can be detected and predicted immediately.

Sammanfattning

Att lära sig känna igen och förutsäga vanliga aktiviteter genom att analysera sensordata från observerade objekt är ett viktigt och utmanande problem relaterat både till artificiell intelligens och robotik. I det här exjobbet studerar vi det generella problemet rörande adaptiv situationsmedvetenhet, och vi argumenterar för behovet av ett angreppssätt som arbetar on-line (direkt på ny data) och från botten upp. Vi föreslår en möjlig lösning som vi anser bereder väg för framtida utökningar som kan ta oss närmare detta mål. Vi presenterar en ny metod för adaptiv aktivitetsinlärning, där en mappning mellan rå-data och grundläggande aktivitetskoncept, samt deras kausala relationer, lärs och är kontinuerligt förfinade utan behov av övervakning. Tillvägagångssättet bygger på användandet av strömmar av observationer av objekt, och inlärning sker av en statistik representation för både de observerade spatio-temporala aktiviteterna och deras kausala relationer. Aktiviteternas dynamik modelleras med hjälp av glesa Gaussiska processer och för att modellera aktiviteternas kausala samband används probabilistiska grafer. Givet observationer av ett objekt så stödjer de inlärda modellerna både skattning av den troligaste aktiviteten och förutsägelser av de mest troliga framtida (och dåtida) aktiviteterna utförda. Metoder och idéer från en rad olika tidigare arbeten kombineras på ett sätt som möjliggör ett enhetligt och effektivt sätt att hantera en mängd vanliga problem relaterade till inlärning, klassificering och förutsägelser av aktiviteter. Ramverket är utvärderat genom att dels inlärning av aktiviteter i en simulerad trafikövervakningsapplikation och dels genom inlärning av flygmönster hos ett internt utvecklad quadrocoptersystem. Slutsatsen är att vårt ramverk klarar av att lära sig de observerade aktivisterna i realtid med god noggrannhet. Vi ser detta arbete som ett steg mot oövervakad inlärning av aktiviteter för robotsystem, så att dessa kan anpassa sig till nya förhållanden autonomt och lära sig nya aktiviteter direkt och som då dessutom kan börja detekteras och förutsägas omedelbart.

Contents

Acknowledgments

I have always been interested in robots and artificial intelligence, in one form or another, ever since I can first remember. The older I became, the more interesting and fabulous the details and concepts were, while I at the same time grew in disappointment of how little progress it seemed we had made. I could imagine such potential, and yet it felt like we were lifetimes away from getting there. I no longer feel as disappointed, nor am I as worried. We are slowly but steadily getting there, a little closer and a little faster each year.

Since many years back now, I have come to appreciate what effect growing up in my generation has had on me. Personal computers, and later the Internet, formed a natural part our daily experience as teenagers and forward, with Google and Wikipedia at a few pushes of a button away. What older generations still see as recent wonders, especially engineers and in academia, we grew up with and never really had a chance to appreciate its glamour in the same way.

I have a desire to replicate and automate my skills and how I go about learning and applying knowledge for solving problems and acquiring new knowledge. One of the fundamental aspects of a learning entity is the ability to distinguish patterns of increasing detail, such as when we explore something new and after a while start to see and understand more and more details about it. This is for example the case with music of a new genre, or that of whiskey or wine, were what is new first can seem awful and weird. However, after some accumulated experience it is shown to consist of many different flavors, with intricate details one didn't think were there. It is something I have found to be the case with most things in life.

In this work I present a way to go about this incremental learning of details, currently applied to that of spatial behaviors, with the sense-reasoning gap as one of the main hindrance for the progress of robotics and AI. I look forward to seeing where it can get us and I am eager to continue, on the shoulders of giants, the improvements of civilization that future generations one day will come to take for granted.

Many people have made this work possible with both minor and major contributions of various sorts. I would first like to thank my examiner Fredrik Heintz for his engagement, encouragement and support. I have the pleasure of working alongside him and my supervisor Daniel de Leng on a daily basis, whom I would like thank for all the enlightening discussions, his tireless support and for always being there to help me. I would also like to thank everybody at AIICS, and Patrick Doherty in particular, for welcoming me to the division. A special thanks to Patrick Doherty for the generous opportunity to present my preliminary work at the STAIRS symposium of the 21th European conference on artificial intelligence (ECAI). It taught me a lot to attend the conference and to meet my peers, with interesting discussions and many new contacts as a result.

Finally I would like to thank my family and friends for their support and en-

couragement during these past six years of study. I am every bit as proud to be their family and friend as they might be for my success and the conclusion of this chapter of my life. Lastly an especial thank you to my extraordinary and beloved Hanna Johansson, for our past years and those yet to come.

Linköping, October 2014
Mattias Tiger

1 Introduction

This chapter starts with an introduction to the problem of situation awareness in a realistic and dynamic environment. It is followed by an introduction to a novel and efficient approach to this problem using unsupervised bottom-up learning of activities. The objectives of this thesis are covered together with its contribution, and finally a thesis outline is given in which the remaining chapters are introduced.

1.1 Introduction

One of the core issues in Artificial Intelligence is the notion of *situation awareness*, introduced by Endsley (1988) as "knowing what is going on around you" in a given situation. Concretely, what we seek is adaptive domain knowledge acquisition through continuous maintenance and integration of high-level and low-level information combining symbolic and quantitative models. This brings us closer to learning, recognition, prediction and simulation of situations on all abstraction levels, which is closely related to efforts towards bridging the *sense-reasoning gap*, introduced by Heintz et al. (2010).

Suppose you want to build the mind of a robot that interacts with and performs human level tasks in the real world. In a lab the environment can to a large extent be controlled and behaviors can therefore be hard-coded for different scenarios given the known artificial constraints. Real world environments are in many cases much more challenging by being more dynamic and varying than those in the controlled lab setting, and would require a lot more hard coded functionality to manage each expected situation for each task. It can be very hard to pin down how the real world environment might change and to anticipate all appropriate

responses and situations to detect and adapt to.

Should you let the robot run free in the environment and something unanticipated were to happen, then the expected behavior is undefined and anything can happen. In some cases it might be possible to detect when the robot's world model diverges from the real world in a way that can make it stop in unknown situations and wait until everything is as it is used to, before carrying on. This would definitely improve the safety of humans and property, similar to how an industrial robot might stop when a human or something noticeable enters a restricted space as to not hurt the human. But permanent or long changes to the environment will lead to the robot standing idle a very long time or even indefinitely, unless humans intervene.

The changes in the environment might be either slow or very rapid and models used by the robot can slowly drift from the piece of reality it models or become deprecated all together. Unless total knowledge of a domain is available from the start, an operator would have to figure out a new set of rules with corresponding context and update the robot each time it detects a limitation. Maybe we will have acquired sufficiently complete domain knowledge models for real world domains in the future, but we are no way near that point yet, so it is still up to the robot developer to figure out abstractions, rules, contexts and restrictions for their robot.

Another way to approach this problem is to add mechanisms that extends and updates the robot world model. While it is comparatively simple to update some quantitative models of e.g. probability distributions or certain classifiers, it is much harder to extend a robot's taxonomy with new concepts that are also at the same time grounded/connected from the bottom-up in everything else that the robot understands. If a robot cannot relate a newly learned concept to the other concepts it knows then it cannot reason over its effects. If it cannot relate the new concept to some quantitative representations then it cannot simulate its effects in the continuous world, nor keep it updated or maybe not even detect it properly. The integration of low-level and high-level data representations and processes is one of the main challenges of autonomous embedded systems (Coradeschi, 1999).

A way to approach these issues is to let the robot itself learn the domain knowledge in an unsupervised manner, from low-level representations up to high-level concepts and relations, bottom-up from observation. The idea is that what has been learned can also be relearned, modified, kept updated and will continuously in time represent the most accurate up-to-date world model the robot is capable of. **The objective of this thesis is to present a novel approach to adaptive activity and context learning, where a mapping between raw data and primitive activity concepts and contexts are learned and continuously improved without supervision.**

Activities here are so called spatio-temporal activities, which are generally defined as changes in state variables over time. State variables can for example be from human activities in the form of arm or leg motions related to the torso, or

related to the other arm or leg. In a traffic domain state variables can be the position and velocity of a driving car, or come from relations between cars in for example a crossing or during an overtake maneuver. Examples of car activities can be a left turn, a slowdown, acceleration, a parking maneuver or an overtake, and each of them of various kinds. It is also important to recognize that activities do not happen in isolation. Activities follow after each other, are triggered by previous activities and are causing future activities. Activities differ at different places, at different times and in different situations. The recognition and prediction of activities is dependent on the context, to narrow down an overwhelming set of possibilities what is most likely to be applicable and most expected, or as Brézillon (1999) put it: "Contextual knowledge acts as a filter that defines, at a given time, what knowledge pieces must be taken into account (explicit context) from those that are not necessary or already shared (implicit context)." *Context* in this work is information external to the spatio-temporal activity models and can be things like causal transitions between activities, object classes performing specific activities or time periods and time cycles of variations in behavior. The notion also incorporates discrete information such as the presence of a red light for a car or rain for a UAV.

We focus primarily on the case of streaming data, where observations are made incrementally available. We assume that we cannot wait until everything worth observing has been observed, and therefore require an incremental build-up of domain knowledge through learning. The reason for this is that we want to be able to use real-time adaptive learning systems in the field, on-board robotic platforms or in surveillance applications. It is also the case that the number of possible situations in the real world is practically endless and the situations ever changing, so we will never have learned everything with a complete and correct representation of the world. Just like the world is changing, so must we.

1.2 Problem Overview

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 that a person has forgotten to take her medicine in an assisted home (Aggarwal and Ryoo, 2011), monitoring telecommunication networks (Dousson and Maigat, 2007), and recognizing illegal or dangerous traffic behavior (Gerber and Nagel, 2008, Heintz et al., 2007). In each of the applications the input is a potentially large number of streams of observations coming from sensors and other information sources. Based on these observations common patterns should be learned and later recognized, in real-time as new information becomes available.

The road traffic domain is an example of a domain which is composed of explicit trajectories, consisting of states that change over time. The domain of trajectories of positions of cars along roads is simple enough to be used as an illustrative example, yet rich enough in complexity and applicability. Changes in state vari-

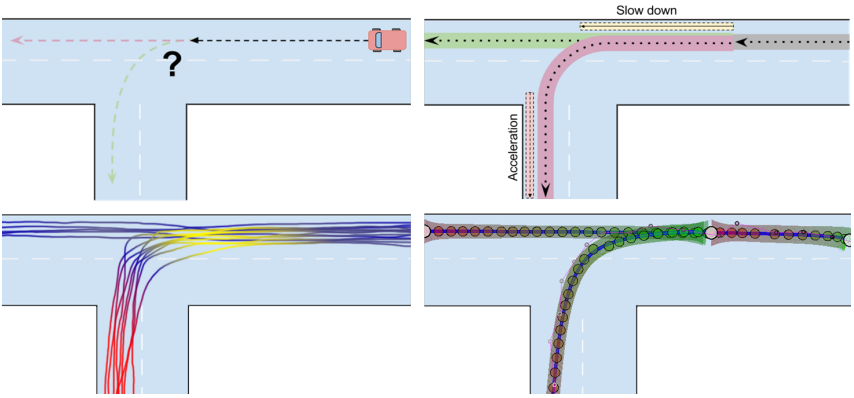


Figure 1.1: *T-crossing scenario. Will the car turn or continue?*

ables (motion) are constrained by environmental structure (roads) which enforce a context very well suited for trajectories. Such a context would in many other domains have to be learned first. This would require a larger system than what we focus on here, although we give some hints about the larger perspective. By choosing this domain we can focus on the fundamentals of quantitative to qualitative domain knowledge learning together with a base context layer, and save the next layer of in-depth context learning for future work. The work in this thesis is with the latter extension in mind.

Consider a traffic monitoring application where a UAV is given the task of monitoring the traffic in a T-crossing. The UAV is equipped with a color and a thermal camera and is capable of detecting and tracking vehicles driving through the intersection (Heintz et al., 2007). 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 1.1, left). Based on this information the UAV needs to learn traffic activities such as driving straight through the intersection or making a turn (Figure 1.1, right). The T-crossing scenario in Figure 1.1 is used throughout the thesis as a running example.

The learning of activity models is similar to trajectory clustering since the model represents a (growing) set of observed trajectories. Morris and Trivedi (2013) highlights two key questions in trajectory clustering which are as follows:

- "How can trajectories of varying length be compared (clustered) in a manner that ensures all semantically meaningful patterns are extracted in a completely unsupervised fashion?"
- "Given a grouping of similar trajectories, how should they be modeled and parametrized to capture the difference between clusters while providing a computationally efficient inference scheme?"

The first question is covered in-depth in chapter 5 where we present our unsupervised framework aimed to accomplish this task. An intuition of our solution to the first question is also provided in section 4.2. The second question is covered in chapter 3 and 4, where we provide a computationally efficient probabilistic model that we show in chapter 6 can be used for both on-line unsupervised learning in real-time, and be used for classification and prediction in real-time.

1.3 Objectives

The objective is to construct and evaluate an unsupervised domain knowledge acquisition framework for spatio-temporal activities. The domain knowledge base constructed should be suitable for activity learning, recognition, prediction and simulation on multiple abstraction levels. The abstraction levels focused on in this thesis are the two which are considered to be fundamental for spatio-temporal activities:

- Learn and update atomic state-trajectories using Gaussian processes as a probabilistic continuous model. These will be our atomic building blocks and form the instances of atomic/primitives activities. The models should be limited in size so that they do not grow with additional observations, and the learning mechanism should make sure of this. Learning, updating and comparison regarding similarity with other activities or trajectories should be as computationally efficient as possible in order to meet the real-time constraint.
- Segment overlapping atomic activities and learn the proximity and transitional relations between activities in a way that makes it possible to perform probabilistic prediction, novelty/abnormality discovery and transition detection. We wish to capture the contextual information that is directly available from comparing new trajectories to known activity trajectories regarding their overlap in the state-space and that they have a direction in time.

A number of properties are desired of a solution of the stated problems. These are highlighted with the notation **[D]**. We will use the properties when evaluating our framework.

Given incrementally available state trajectories of unique objects, we want to learn **[D1]** and maintain **[D2]** an up to date knowledge base of the world. The state variables are assumed to be continuous, such as physical properties which are sampled by sensors. The intention is to have such a system in the future running on a robotic platform capable of acting in the real world **[D3]**. This means that real-time restrictions and information overload must be taken into consideration. The real-time constraint is important since a robot acting in the real world has a need to react on time to changes in its surroundings. The issue of information overload is important since a continuously running system will observe immense amounts of data, which can be useful in modeling and understanding the world. The ideal is a system that can run indefinitely in real-time **[D4]** and

accumulate an endless number of observations **[D5]**, allowing autonomous operations with adaptive world understanding support. We assume the presence of robust tracking and that the observed trajectories are produced by the intended objects, although the trajectories and each single observation are allowed, and expected to, contain observation noise. We assume this noise has a Gaussian distribution.

The knowledge base that is learned should be useful for a robotic platform or surveillance system. It should support recognition **[D6]**, prediction **[D7]** and simulation **[D8]** of activities and situations, thereby providing situation awareness (Endsley, 1988). World understanding encompasses the ability to recognize behaviors and situations as well as the ability to reason over causal relationships **[D9]** between different behaviors and situations. The causality used here is probabilistic causality, which means that a behavior B follows behavior A with probability P, for example. We want a system to be able to recognize previously learned behaviors as well as observed behaviors that are unknown to the system **[D10]**. We also want it to recognize possible causes for the observed behavior **[D11]** and to which degree this behavior is explained or considered unknown given what has been seen and learned so far.

We want to learn activities without supervision **[D12]**, both to simplify for humans but also to let robotic systems adapt to new circumstances, to learn new activities on the fly that can be detected and predicted immediately. We also want to provide up to date certainty measurements of detected and predicted activities, for event processing and other high level stream reasoning and Artificial Intelligence (AI) applications.

1.4 Contributions

We have constructed an activity learning, recognition and prediction framework (Tiger and Heintz, 2014) that was presented as a poster at the STAIRS symposium of the 21th European conference on artificial intelligence (ECAI). It provides the fundamentals for a system with capabilities of the sort discussed in the beginning of this chapter. These fundamentals have been out-lined and evaluated in the thesis. We show a way to model and learn atomic state-trajectories using Gaussian processes and demonstrate that this approach has some key properties:

- Fully probabilistic framework on all abstraction levels.
- On-line unsupervised learning without unjustified model growth.
- One-shot learning: a single observation is sufficient for recognition, prediction and simulation of an activity or activity chain.
- Real-time performance.
- Convergence under realistic assumptions.
- Scalability to real-world situations and to complex context learning.

The framework is demonstrated to be able to learn, recognize and predict spatio-temporal activities in real-time, both in simulations and in practice.

Methods and ideas from a wide range of previous work are combined and interact to provide a uniform way to tackle a variety of common problems related to learning, classifying and predicting activities. We provide an extensive approach towards the modeling and realization of real-time context-sensitive adaptive situation awareness.

1.5 Outline

The thesis consists of two main parts, with a background chapter as prelude. The first part consists of chapter 3 and 4. It concentrates on how to model activities as atomic state-trajectories using unsupervised methods in a manner that is both efficient enough for real-time usage and adaptive, but with a size that does not grow with an increasingly number of observations. The second part consists of chapter 5 which uses the atomic activity model introduced in part one to learn, segment, recognize and predict atomic activities. It concentrates on how the relations between atomic activities are learned, a presentation of the entire framework and concludes with how it can be used for recognition, prediction and detection tasks.

The thesis is structured as follows:

- Chapter 2 covers background information about streams, stream processing and Gaussian processes which is used in the remainder of the thesis.
- Chapter 3 introduces local and sparse Gaussian processes and we propose an approximation schema based on sampling Gaussian processes that provide a fixed size trajectory model suiting our needs.
- Chapter 4 introduces spatio-temporal activities and the activity model we use. Aspects such as evidence and similarity between activities are presented.
- Chapter 5 presents the activity learning and classifying framework and proposes an activity segmentation scheme for learning probabilistic causal transitions and relations between activities, enabling prediction and abnormality detection among other things.
- Chapter 6 describes experiments for two different scenarios, using simulated and real data.
- Chapter 7 concludes the thesis with a discussion and conclusions.

2

Background and Related Work

In this chapter we introduce the stream concept and its relation to robotics and signal processing. The added value of having up to date certainties of the constraints used in symbolic reasoning, although not reasoned over explicitly, is illustrated and used as a motivational example of the bridging of sub-symbolic and symbolic knowledge representations. Velocities of an object are measured as discrete sample values which are point-wise estimates of a velocity fluent that has a continuous form in time. Gaussian processes, which are continuous probabilistic non-parametric models are introduced in order to have a model that can capture the underlying fluent of properties, such as velocities, from discrete sampled observations.

2.1 Stream Processing and Stream Reasoning

A stream of information is characterized by information being made available incrementally. Most of todays information exchanges are stream-based by the use of serial interfaces. The information transferred as a stream can be divided into two distinct types based on intent. The first type is information that is only valuable if it is recent enough. The second type is information as data structures which are only valuable when they are fully transferred. The information from a sensor in an active robotic system can be of the former type, and of the latter type if meant to be part of a data set with a clear beginning and end. Similarly a video file or a movie can be both, since it is intrinsically stream-based in that it contains an ordered data structure of frame after frame from a start to an end. In comparison, a binary program file is in most cases useless in parts, with no linear ordering of the internal structure, and therefore requires the transfer to be complete before it becomes useful. In robotic systems streams of information are

mostly of the first type. In a changing world a robotic system is desired to be able to react and adapt to the changes it faces *when* they happen.

In this thesis we are primarily concerned with streams originating from measurements of the physical world. The measurements can for example be of properties associated with objects or properties associated with relations of objects. The values of these properties can be continuous and changing over time and the total function from time to value of such a property is called a fluent (Heintz et al., 2007). A sensor produces a fluent stream, a partial representation of a fluent, by providing a momentary estimate of a fluent's value for each time point a sensor read-out is made. The estimate varies in quality due to the imperfectness of any physical setup. It is possible to model this imperfectness with a probability distribution such as a Gaussian distribution. By using for example a known or measured variations in estimated quality as a variance estimate and the sensor value as the distribution mean, a Gaussian distribution is used as a model for the imperfectness of the sensor itself.

When both the an estimate and its uncertainty are made available, probabilistic inference such as (sensor) fusion is made possible. Virtual filters can be constructed using Bayesian inference schemes, such as the Kalman filter (KF) or the more general Particle filter. These are used to estimate fluent streams of not directly observable fluents by the use of physical models connecting the unobserved property with observed properties. While an ordinary sensor in many cases can be assumed to have a static probability distribution over its measurement, the virtual filter usually adapts its estimates and associated probability distributions with each new measurement. A sensor might measure the velocity of an object, and a Kalman filter might use this to additionally estimate the position and the acceleration of the object. For the sensor each measure estimate is regarded as independent of the measurements in the past or future. In the case of the Kalman filter, the estimated distributions at any given time is the result of the effects of the previous state, the current received measurements, as well as predicted or expected values provided by the physical model. This combination of the past, the present and the expected present can make it possible for the Kalman filter to become more certain of an estimated property, say velocity, than the sensor measuring the property in question. In other words, the fluent stream provided by a virtual sensor might be of a higher quality in its estimate of the fluent than the sensor which produce the initial fluent stream of the fluent in question.

Cugola and Margara (2012) divide stream processing systems into two main categories, Data Stream Management Systems (DSMS) and Complex Event Processing (CEP). The former is an adaptation of the traditional Data Base Management System (DBMS). The DSMS model work with streams instead of with persistently stored data and execute standing queries which run and provide answers continuously as new data arrives in the input streams. Streams are transformed into other streams using selections, aggregations, joints and other transformations based on common SQL operators and defined by relation algebra. In contrast,

the CEP model consider stream items as event notifications (e.g. of events in the external world). These are filtered and combined to find patterns that match those of high-level events, which reflect what is happening in the external world. It is the process of abstraction, by detection of high-level events from patterns of low-level events, that is central to the CEP model.

The properties of stream transformations (DSMS) and stream abstractions (CEP) are required to have full-fledged Information Flow Processing (IFP), a general term for stream processing systems. Except for the unique properties of respectively DSMS and CEP, the central aspect of IFP systems is "the need for processing information as it flows from the periphery to the center of the system without requiring, at least in principle, the information to be persistently stored. Once the flowing data has been processed, thereby producing new information, it can be discarded while the newly produced information leaves the system as output" Cugola and Margara (2012).

An example of a IFP system is DyKnow (Heintz, 2009, Heintz and Doherty, 2004), a stream-based knowledge processing middleware framework. They use *Metric temporal logic* (MTL), an extension of first order logic with the addition of temporal operators, to provide stream reasoning over streams. *Stream reasoning* is incremental reasoning over incrementally available information. It makes DyKnow able to react to a rapidly changing environment due to the minimal latency in the incremental formula evaluation by a technique called progression. CEP is included in DyKnow by the incorporation of Chronicle Recognition (Dousson and Le Maigat, 2007) which provides CEP-specific functionality. DyKnow has also been extended with a formal model and semantic matching. The latter makes streams of indirectly-available information available by dynamically constructing the necessary transformations (from transformations that are available) to produce the requested streams (de Leng, 2013).

2.2 Motivation for Using Probabilities

Sensors and virtual sensors can be seen as measuring probability distributions and in practice it is most often the mean which is used for (constraint) comparison in cases where uncertainties or probability distributions are not taken into account. This is also the case in metric temporal logic (MTL) reasoning and in chronicle recognition. The form of the probability distributions does however impact the uncertainty of that value, and the availability of this uncertainty can enable better decision-making. Consider the usage of $Altitude > 100m$ as a constraint for an event. In this example the altitude sensor can measure the altitude property with a Gaussian probability distribution. If the first moment (the mean) of the distribution is sampled as 101m, the constraint is satisfied. The likelihood for the altitude to actually be above 100m does however depend on the second moment (the standard deviation) of the Gaussian distribution, as is illustrated in Figure 2.1.

If the standard deviation is 5m, the probability of the altitude being higher than

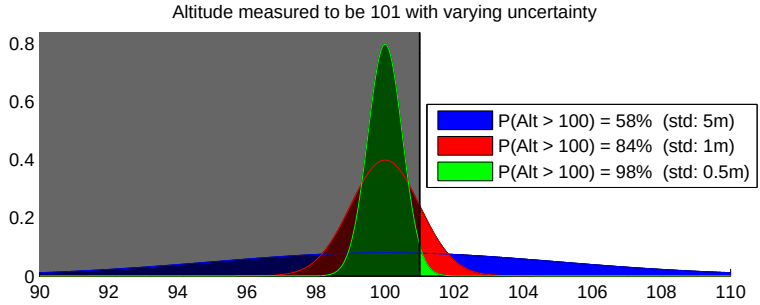


Figure 2.1: *Illustration of the difference in the certainty that the altitude (Alt) is above 100m given various degrees of uncertainties in the sensor value for altitude with a measured altitude of 101m.*

100m is only 58%. If the standard deviation is 1m the probability is 84%. If a 98% certainty of the constraint being satisfied is desired, the altitude constraint can easily once and for all be adjusted upwards to compensate for the uncertainty if the probability distribution is static and known. However, sensors may be replaced or upgraded, or the software might be meant to run on more than one system. By letting the system know the probability distribution of the sensor, it is possible to simply state the degree of certainty that is desired and the system could take care of the rest. Similarly, virtual sensors such as Kalman filters have probability distributions which change over time depending on how well the models used reflect the observed data, updating the uncertainty of the estimated value for every sample. The managing of dynamic certainty introduces robustness in the system by mitigating faulty perception, so that the constraints will be satisfied with the same desired certainty regardless how uncertain the agent has become. This can be due to an insufficient model, or due to degradation of sensors as long as the sensor fault can be detected and estimated. It makes the system adaptive to increases and decreases in the quality of perception without any change of policies. For example, a UAV will fly as high as necessary in order to fulfill the required certainty that it is at least above a threshold altitude, regardless of perception quality.

While it is straightforward how to handle the certainty of constraints based on physical attributes from sensors or virtual sensors, it is more difficult when it comes to the detection of activities. An activity can informally be defined as certain changes of some physical attributes over time. Just like a KF is used to estimate the true value of a fluent at a given time point, an activity as a model is an estimate of some fluents over a time frame. An activity can be a human's action of flexing an arm or a car performing a turn or a slowdown. If a car slows down or accelerates the activity model may estimate the fluent, the positive or negative function of a cars acceleration over time. Since such an activity is a state change in a continuous space it is beneficial to model it using a continuous probabilistic model, to sufficiently expressively model the fluents behind the observed

uncertain samples. Consider that we want to describe the kind of slowdown a car performs in a T-crossing before it turns. This would be very hard to specify formally, making a good argument for learning. To achieve this without learning it would be necessary to observe a great number of similar car-paths, as well as complementary paths to discriminate with. Then a statistical model has to be constructed of all those observations which best explains the dynamic changes over space, time and acceleration. This would indeed be what a learning algorithm would do, either unsupervised or supervised.

2.3 Gaussian Processes

Gaussian Processes (GPs) have been shown to be useful for prediction, modeling and detecting spatio-temporal trajectories such as motor vehicles in crossings (Kim et al., 2011), marine vessel paths (Smith et al., 2013) and human body dynamics (Wang et al., 2008). GPs have also been used to model spatial regions and structures such as occupancy maps (Kim and Kim, 2013b) and smooth motion paths (Keat and Laugier, 2007). They are also good for handling noisy or missing data (Kim et al., 2011, Wang et al., 2008), where large chunks of trajectories can reliably be reconstructed.

A Gaussian process is a distribution over functions,

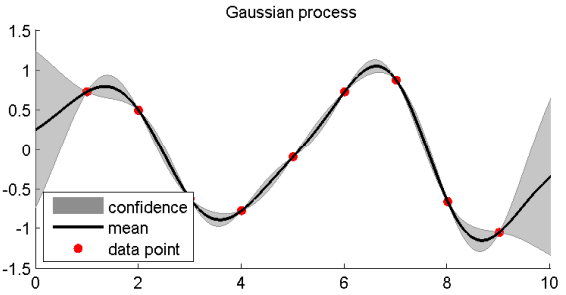
$$f(x) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x})), \tag{2.1}$$

fully specified by its expected mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x})$ (Rasmussen, 2006). It is a generalization of the multidimensional Gaussian distribution which in turn is a distribution over vectors,

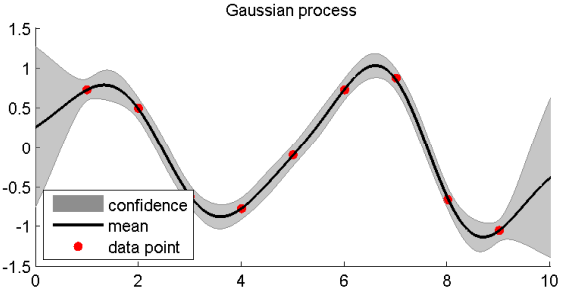
$$\mathbf{x} \sim \mathcal{N}(\mu, \Sigma), \tag{2.2}$$

fully specified by its mean vector μ and covariance matrix Σ. The stochastic vector $\mathbf{x}$ in the multidimensional Gaussian distribution is usually used to model a finite set of dimensions, such as a 3D position or the angle of each joint in a robot. For the Gaussian process it is the function values $\mathbf{y}$ and arguments $\mathbf{x}$ ($y = f(x)$, $y \in \mathbf{y}$, $x \in \mathbf{x}$) to the stochastic function that are each a stochastic vector, a vector not over the dimension of the input respectively output, but over the domain and co-domain, respectively, of the function. Although the domain of the function may be the real numbers, $\mathbf{x}$ will in application only consist of a finite subset of the possible inputs and $\mathbf{y}$ of the corresponding outputs. These function arguments and function values $(\mathbf{x}, \mathbf{y})$ will be those that are known, in this work referred to as *support points* of a *Gaussian process model*, and which provide the means of interpolating and extrapolating the stochastic function for other input values. $m(\mathbf{x})$ is a vector and $k(\mathbf{x}, \mathbf{x})$ is a symmetric matrix, similar to μ and Σ of the multidimensional Gaussian distribution. A Gaussian process can therefore be seen as a multidimensional Gaussian distribution over its known input values. A Gaussian process is a set of jointly Gaussian stochastic variables; $\mathbf{x}$ and $\mathbf{y}$.

In many applications it is common that the known outputs, which are often based



(a) Without noisy outputs.



(b) With noisy outputs.

Figure 2.2: *Example of two Gaussian process distributions, with a zero mean function and a Gaussian kernel as covariance function. The grey area is the 95% confidence interval. The red dots are the means of the instances of the stochastic variable pairs $(x, f(x) = y)$ of the finite set this Gaussian process model consists of.*

on measurements, contain noise. A common assumption is that the noise in the output is independent and Gaussian distributed. Since the noise is independent the only effect it has on the covariance is that on the autocovariance of each stochastic variable $f(x)$. This formulation takes the form

$$\hat{f}(x) = f(x) + \epsilon(x), \quad \epsilon(x) \sim \mathcal{N}(0, \sigma^2_{noise}(x)) \tag{2.3}$$

$$\hat{f}(x) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}) + \sigma_i^2 \delta_{ij}), \tag{2.4}$$

where i and j are the indexes in $k(x_i, x_j)$, and σ_i^2 is the output variance of $y_i = \hat{f}(x_i)$ for each $x_i \in \mathbf{x}$ and $y_i \in \mathbf{y}$. $\delta_{ij} = 1$ if $i = j$ else, zero. The evaluation of the stochastic function $\hat{f}$ for the argument x^*,

$$y^* = \hat{f}(x^*), \tag{2.5}$$

is a Gaussian distributed stochastic variable y^* with the conditional distribution

$y^* \sim p_{\hat{f}}(x^* \mid \mathbf{x}, \mathbf{y}) = \mathcal{N}(\mu(x^*), \sigma^2(x^*))$, where

$$\mu(x^*) = m(\mathbf{x}) + k(x^*, \mathbf{x})(k(\mathbf{x}, \mathbf{x}) + \sigma_i^2 \delta_{ij})^{-1}(\mathbf{y} - m(\mathbf{x})), \tag{2.6}$$

$$\sigma^2(x^*) = k(x^*, x^*) - k(x^*, \mathbf{x})(k(\mathbf{x}, \mathbf{x}) + \sigma_i^2 \delta_{ij})^{-1} k(x^*, \mathbf{x})^T. \tag{2.7}$$

Here $k(x^*, x^*)$ is a scalar, $k(x^*, \mathbf{x})$ a N-dimensional row-vector and $k(\mathbf{x}, \mathbf{x})$ a NxN-dimensional matrix, where N is the number of elements in respective vectors $\mathbf{x}$ and $\mathbf{y}$. From here on we refer to $\mu(x^*)$ from equation 2.6 as the *mean function* and $\sigma^2(x^*)$ from equation 2.7 as the *variance function* of a Gaussian process model.

Some degree of noise in the inputs is also common in practice, but it is usually ignored, as it is much harder to handle than noise in the outputs. There exist approximate solutions to noisy inputs (Girard et al., 2002). An example of a a Gaussian process with and without observation noise is shown in Figure 2.2.

The above formulation is easily extended to handle multiple output variables (higher dimension than 1 in y) if these are assumed independent. If the outputs have the same variance, the only change is that the mean function now is M dimensional and $\mathbf{y}$ is $M \times N$ dimensional, where M is the dimension of y and N is the number of stochastic variable pairs in the Gaussian process model. Extensions exist to handle multiple dependent and identically distributed output (Boyle and Frean, 2005).

If the input consists of multiple variables instead, the GP turns into a Gaussian random field. Given that the kernel function is a mapping from the higher input space to a scalar space, e.g. by being a function of the norm of the input, then it can be treated as a ordinary GP.

One major obstacle restricting the use of GPs in the past has been the expensive matrix inversion operation necessary to update the model. A straight forward implementation of a Gaussian process require the inversion of the covariance matrix, an operation with the computational complexity of $\mathcal{O}(n^3)$ in the size of the training data. Various methods have been employed to reduce this weakness, by efficient iterative updates (Smith et al., 2012), segmentation into local patches of GPs (Schneider and Ertel, 2010, Snelson and Ghahramani, 2007), and by techniques to make the GP sparse (Snelson and Ghahramani, 2005, 2007). In this thesis we employ sparse GPs to model activities, which are systematically segmented into local models. We extend the Gaussian process regression to also regress sparsely over the number of known inputs and outputs, in a way that better suits our applications.

2.3.1 Kernels

It is common to assume that the mean of a Gaussian process is zero for many calculations in practice. This would mean that the mean function is zero. In our application we do not assume that the data of the GP is normalized to be centered around zero and we therefore assume the mean function to be a constant with the value of the mean of the output values $\mathbf{y}$ of the GP model.

We make use of the squared exponential kernel which is commonly used in Gaussian Process regression,

$$k(x_1, x_2) = \theta_0^2 e^{-\frac{\|x_1 - x_2\|_2^2}{2\theta_1^2}}, \tag{2.8}$$

where θ_0^2 denotes the global variance of the mapping and θ_1^2 denotes the global smoothness parameter of the mapping. It is a non-linear mapping which allow a Gaussian process to model arbitrary linear and non-linear functions.

The smoothness term in equation 2.8 is equal for all D dimensions of x. However, different dimensions might be suitable to be smoother than other. It is also the case that some dimensions might be dependent, and this formulation assumes independence for all dimensions of $x \in \mathbb{R}^D$. A more general formulation is

$$k(x_1, x_2) = \theta_0^2 e^{-(x_1 - x_2)^T W (x_1 - x_2)}, \tag{2.9}$$

where W is a DxD matrix with three different interesting possibilities:

$$W = \theta_1^2 \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix} \tag{2.10}$$

$$W = \begin{bmatrix} \theta_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \theta_D^2 \end{bmatrix} \tag{2.11}$$

$$W = \begin{bmatrix} \theta_{1,1}^2 & \dots & \theta_{1,D}^2 \\ \vdots & \ddots & \vdots \\ \theta_{D,1}^2 & \dots & \theta_{D,D}^2 \end{bmatrix} \tag{2.12}$$

The first possibility (equation 2.10) makes the kernel equivalent to equation 2.8. The second possibility (equation 2.11) is slower to optimize (D smooth parameters instead of *one*) but it provides a smoothness parameter for each dimension. By utilizing that the matrix W is diagonal, the complexity of the evaluation of the kernel is linear in the number of dimensions D. The third and last possibility (equation 2.12) provides the expressibility of dependent relations between the dimensions, but as a consequence also increases the complexity of the evaluation of the kernel to be quadratic in the number of dimensions D. The parameters in the matrices can either be learned, as part of the Gaussian process, or hard coded by a user. For example, the dependencies of different dimensions might be known a priori, but the smoothness of each dimension that best describes the data in the Gaussian process model might have to be learned.

2.3.2 Gaussian Process Regression

Given a set of data points $\mathbf{x}$ and $\mathbf{y}$, we want to find the hyper parameters θ_0 and θ_1 that make the Gaussian process fit the data in some optimal way. The

hyper parameters are usually optimized over the marginal likelihood $P(\mathbf{y}|\mathbf{x})$ of the Gaussian process, where $\mathbf{x}$ has been integrated out of the likelihood. Gaussian process regression is described in more detail in the work of Rasmussen (2006).

The marginal log likelihood of a Gaussian process which we want to maximize is, as derived by Rasmussen (2006),

$$\log P(\mathbf{y}|\mathbf{x}) = -\frac{1}{2}\mathbf{y}[V]^{-1}\mathbf{y}^T - \frac{1}{2}\log(\det(V)) - \frac{1}{2}N\log(2\pi), \tag{2.13}$$

where $V = k(\mathbf{x}, \mathbf{x}) + \sigma_i^2\delta_{ij}$. The hyper parameters θ_0 and θ_1 that maximize the right hand side set to zero can be solved on closed form.

2.4 Related Work

There are many different approaches to activity recognition, both qualitative and quantitative. Here we focus on some quantitative approaches since they are most related to our work. As far as we know there is no other unsupervised framework that can learn the atomic activities, the dynamics of each activity and the causal relations between the activities in a fully continuous and probabilistic model.

In the field of *vehicular traffic behavior understanding* two of the most popular methods are *trajectory clustering* and *topic modeling* (Morris and Trivedi, 2013). The former uses trajectories of image coordinates as input and is therefore similar to our work. The usage of trajectories make the models dependent on robustness of the tracker, and the latter group of methods do not assume trajectories and instead directly use motion feature vectors from the raw image feed.

One example of the latter approach uses a Bayesian Hierarchical Model with moving pixels from video surveillance as input Wang et al. (2007). They divide the image into square regions which the pixels belong to and use together with four different motion directions as a code book. They treat 10-second video clips as documents and the moving pixels as words and learn a Hierarchical Dirichlet Process. By performing word document analysis, the resulting topics are normative atomic activities in the form of pixel motion trajectories in the 2D image. They can unsupervised learn what they call 'atomic activities', which are activity regions which are separated by lack of motion (i.e. a car or pedestrian stopping). They also learn 'interactions' between 'atomic activities' which are sets of 'atomic activities' that commonly co-exist in the same video clips (document). Their learning approach is off-line.

Our on-line unsupervised framework in comparison requires trajectories, but can learn more sophisticated primitive activities and relations bottom up from these trajectories. Our primitive activities are continuous in any state-space compared to discrete in the 2D image. Ours are fully probabilistic compared to non-probabilistic histogram-based. Our relations between activities are probabilistic, and segmented based on actual intersecting activities compared to frequent occurring in the same video clips and segmented based on where there is a lack of

pixel-motion. Since our activity relations are actually casual transitions directly acquired from observations, we can do probabilistically correct non-ambiguous predictions of future and past activities. Our framework could use moving pixels as input, but also other types of information such as 3D-positions if available. It also incorporates the uncertainties of observations.

An additional strength of our approach compared to any other with comparable capabilities that we could find, is that it is on-line and is built to operate for arbitrarily long durations and adapting the models on new observed behaviors. Morris and Trivedi (2013) write that adaptation to new data and changing conditions are important for real-world implementations, with the motivation that surveillance systems usually are required to be operational over extensively long time periods and that old training data at some point in time may no longer accurately reflect the current monitoring situation. An example mentioned is road construction shutting down a lane and rerouting traffic. Our framework does not have to remove old behavioral models due to them getting old and more seldom used. This is the case since they will be less and less likely for each new observations that reinforce the evidence of other more frequent behaviors. This provides a natural balance between historical prior models and learning from new observations, as argued for by Morris and Trivedi (2013), and this balance is an implicit feature of the design of the framework with our activity model and activity relations.

Among the trajectory clustering approaches, one approach is to use splines with rectangular envelopes to model trajectories (Makris and Ellis, 2005, Guillarme and Lerouvreur, 2013), in comparison to our approach and the approach of Kim et al. (2011) which use Gaussian Processes in a continuous fully Bayesian setting. Similar to our approach Guillarme and Lerouvreur (2013) divide trajectories into shorter segments and split trajectories at intersections, while Kim et al. (2011) 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 in the learning phase. Our framework can 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.

Piciarelli and Foresti (2006) describe a framework for on-line trajectory clustering for abnormality detection, in which they learn trees of connected wide trajectory-models. The trajectory-models are updated with new observations as to have their width and mean adjusted to new observations. These are created, updated, and merged very similar to our framework. The difference to our work is that our activities are modeled using Gaussian processes, which provide a continuous probabilistic model and that we can handle graphs of trajectories compared to only trees in their work. Additionally updating of activity models we perform rest on a theoretical foundation. Both we and they collect statistics on

how many observed trajectories that have been explained of each modeled trajectory and statistics for the transitions between different modeled trajectories. By using Gaussian processes to model trajectories we make our model much more resilient of missing data. As a consequence it lets us treat the underlying fluent as a continuous function modeled as a Gaussian process distribution which improves the observation quality compared to that of independently observed data points when observation noise is present.

Arguments against trajectory-based behavioral learning of traffic or other outdoor activities include issues with adverse lightning and weather conditions (Morris and Trivedi, 2013) or crowded places (Wang et al., 2007) and occlusion. Spatial trajectories of arbitrary objects from vision sensors that is robust to these kind of effects can with much more ease than previously be made readily available due to recent advances in computer vision of real time robust visual tracking (Danelljan et al., 2014b), (Danelljan et al., 2014a). Robust tracking is therefore no longer very limited in practice for robotic platforms with visual sensors. Trajectories are used by us because they provide additional contextual information that is highly useful when building an understanding of the world bottom-up.

3

Sparse Local Gaussian Processes

Gaussian process model learning is slow for large data sets, with a complexity of $\mathcal{O}(N^3)$ for optimizing the hyper parameters, where N is the number of data points in the data set (Rasmussen, 2006). The complexity for evaluating the mean and variance of a Gaussian process for a single point is $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$ respectively. These complexity constraints has in the past been a major obstacle but since a few years back there exists several approximation techniques which makes Gaussian processes much more appealing and applicable. The two main approximation techniques usually employed are the use of local models and the use of sparse Gaussian processes (Schneider and Ertel, 2010).

The use of local models let us divide the Gaussian process into a chain of smaller GPs, the so called local models, which together are intended to mimic the original GP. This is more efficient than having a single GP since the data points are divided into different models (N is split into a set $\{N_1 ... N_J\}$, where $\sum_{k=0}^{J} N_k = N$). Sparse Gaussian processes use a smaller set of data points as support points compared to all available data points, while still intended to mimic the original GP model. This is more efficient since it reduces the number of data points used (N is replaced by a smaller number M, where it is usually the case that $M << N$).

In this thesis we are faced with a possibly unbounded N, since observations added to the model might continue for an arbitrarily long time period and with arbitrarily high rate. Using the fact that the observations made can be grouped as individual trajectories of finite length over the same ranges, which can be treated as observed incrementally in order, we show that this problem can be solved in an efficient manner.

This chapter starts with an introduction to the problems local models and sparse Gaussian processes solve. It is shown that both of them, individually or combined,

are insufficient for the task at hand with a continuous stream of new observations of trajectories. We introduce an alternative sparse Gaussian process algorithm which we call Sparse Gaussian Process by Uniform Sampling (**SGPUS**) in section 3.4, which builds upon and is motivated by the two preceding sections 3.2 and 3.3. In these sections we show that it is a valid approximation to assume that the Gaussian processes of two observed trajectories can be seen as point-wise partial estimations of common Gaussian distributed populations, and that the joint (combined) estimation of these populations can be modeled point-wise by a new Gaussian process. This is because we assume that both observed trajectories represent instances of the same underlying trajectory which we want to model.

3.1 Sparse Local Gaussian Processes

Local Gaussian process models are used to reduce the complexity of a model by breaking it up into smaller parts that model different pieces of an interval (Schneider and Ertel, 2010). But by breaking them apart the problem of how to move between them arises, as they are no longer connected and can not easily be guaranteed to be smooth and continuously connected. One approach towards a solution to this problem is to let the local models overlap each other slightly, so that bordering models use the same data at their borders (Kim and Kim, 2013a). The neighboring local models will be more similar to each other since they share the same data points at the overlap. Another way to solve the problem is to apply the fusion formula (Gustafsson, 2010) to equation 2.6-2.7 to fuse the information from two or several local models. The result behaves as a single continuous smooth Gaussian process, at the cost of evaluating all active local models for each test point. This cost can be reduced to only that of the two closest local models with a small approximation error if the other models are far away enough, due to the exponential rate of decreasing influence. The fusion formula for two Gaussian distributions is given by equation 3.1-3.2:

$$\sigma_{1,2}^2 = \frac{1}{\sigma_1^{-2} + \sigma_2^{-2}} \tag{3.1}$$

$$\mu_{1,2} = \sigma_{12}^2(\mu_1\sigma_1^{-2} + \mu_2\sigma_2^{-2}) \tag{3.2}$$

where μ_1, μ_2 and σ_1^2, σ_2^2 are the mean and variance respectively for two Gaussian distributions. $\mu_{1,2}$ and $\sigma_{1,2}^2$ are the mean and variance, respectively, of the resulting Gaussian distribution.

The complexity reduction by using local models breaks down when considering examples with very dense or large amounts of data, due to too much data. Such an example is the expected trajectory of cars on an highly traveled road strip, with continuous inclusion of new observations for a few days, weeks or even indefinite monitoring. One straightforward strategy would be to divide a local model into two when the number of data points in its interval reaches a certain size. This however would still make the number of local models rapidly increase. With an increasing number of models, each model will cover a smaller and smaller

interval until it is vanishingly small. The trajectory as a whole would have lost most of the properties of a Gaussian process. This is demonstrated in example 3.1.

3.1 Example

A 100 meter long road segment is being monitored using a static video camera. If cars on the road drive 5 meters apart on average and 40 uniformly distributed car positions per second are sampled (two for each car) then on a month this reaches

$$40 * 60 * 60 * 24 * 30 = 103680000$$

samples in total. If each local model is allowed to hold a maximum of 5000 samples for performance reasons then the average model covers a distance of

$$(100/103680000) * 5000 = 0.0048$$

which is ∼5 mm of the 100 meter long road segment, given a uniform distribution of the observed position in the best case. In many scenarios it would be even worse as two samples per second per vehicle might not provide enough information if the cars are not driving sufficiently slow. For example on a highway two samples per second over a 100 meter road strip would be insufficient for most behaviour recognition applications.

Situations such as these require the local models to be sparse as well, to limit the amount of local models. Sparse Gaussian processes have been used to cope with large data sets and works by either selecting a subset of the data set, or by finding more optimal support points which may not exist explicitly in the data set (Snelson and Ghahramani, 2007). In the example scenario above, the data arrives incrementally and there are many possible applications where a trajectory model might be useful before all the data has been collected. In other scenarios it might also be the case that the number of data points collected becomes too large to store and therefore needs to be incorporated in a sparse model incrementally. The number of data points in example 3.1 is 103680000 for only a 100 m road section after a month. Imagine a city wide statistics collection network.

There exist effective techniques for incremental incorporation of single data points in ordinary Gaussian processes (Osborne, 2010), these are however not known to be trivially extended to the methods that creates sparse Gaussian processes. The support points in the sparse Gaussian processes are each supposedly worth more in influence than a single new observed data point, since they have replaced a potentially huge amount of data points while keeping the shape of Gaussian process.

An alternative way of approaching this problem is to introduce such a weighting mechanism externally of the Gaussian process rather than internally. Treating individual observations of a trajectory as independent is an approximation which disregards the expected underlying trajectory structure of a fluent. It is therefore meaningful to treat such observations as a Gaussian process model which models the underlying trajectory, rather than treating the observations as indi-

vidual Gaussian distributions. This is motivated as long as a sufficient range of observations are used, in order for a Gaussian process to be able to capture the underlying trajectory structure. An example of a trajectory is the observed states of a car driving along a path. A Gaussian process model can be treated as a function $x \in \mathbb{R} \mapsto \mathcal{N}(\mu(x), \sigma^2(x))$ which is a function from the real numbers to a Gaussian distribution parametrized by the functions $\mu(x)$ and $\sigma^2(x)$ from Equation 3.12-3.13. It is possible to calculate a weighted combination of two Gaussian processes by treating them as two pairs each consisting of a mean and a variance function (equation 3.12-3.13). This is shown in section 3.3 which builds upon the content of section 3.2. The weighting can reflect that one Gaussian process model represents a single trajectory and another Gaussian process represents several combined trajectories. We show in section 3.4-1 that by sampling a combining of two GPs we can estimate a new Gaussian process which have the characteristics of the combined GP in the sample interval. The quality of the estimated Gaussian process increases with a higher sample density, although this also increases the size of the model, which in turn makes the Gaussian process model slower. This method provides a way to incrementally include an arbitrary number of observed trajectories into a single model with a size that does not grow because of additional observations.

3.2 Combining Gaussian Distributions

Consider that we draw observations x_k from a Gaussian distribution with unknown mean μ and variance σ^2. We can estimate these parameters by calculating the sample mean and sample variance,

$$\mu_n = \frac{1}{n} \sum_{k=1}^{n} x_k, \tag{3.3}$$

$$\sigma_n^2 = \frac{1}{n-1} \sum_{k=1}^{n} (x_k - \mu_n)^2, \tag{3.4}$$

for all n up-till-now made observations. The sample variance presented here is unbiased and has because of this the dividing factor of $n-1$ instead of n like the sample mean. This is the case because if we only have a single observation the variance is always zero, due to the mean μ_0 and the observation x_0 being the same. So although the sample variance is a sum over all n observations, only $n-1$ observations can be accounted to be used for the variance estimation. As n grows large, both μ_n and σ_n^2 converge to the true values of μ and σ^2, and the bias correcting term makes no difference any more.

Consider instead that each observation contains observation noise such that each observation by itself is modeled by a Gaussian distribution. We no longer have the observed sample x_k but instead have the observation $\mathcal{N}(\mu_k, \sigma_k^2)$. Since x_k is no longer known, we cannot use Equation (3.3-3.4) directly to estimate the true values of μ and σ^2. The observed μ_0 and σ_0^2 form our best estimate of the

unknown Gaussian distribution after the first ($k = 0$) observation. Compared to the previous case a variance is now provided with the first observation, so no bias correction is necessary.

Let $\mathcal{N}(\mu_1, \sigma_1^2)$ and $\mathcal{N}(\mu_2, \sigma_2^2)$ be two sampled estimations of *the same* Gaussian distributed population given N_1 and N_2 observations respectively. In the case of two Gaussian samples from the population then $N_1 = N_2 = 1$. The *combined* sampled estimation is denoted $\mathcal{N}(\mu_{1,2}, \sigma_{1,2}^2)$ with $N = N_1 + N_2$ observations and with $\mu_{1,2}$ and $\sigma_{1,2}^2$ given by

$$\mu_{1,2} = \frac{1}{N_1 + N_2}\left(\sum_{k=1}^{N_1} x_{k,1} + \sum_{k=1}^{N_2} x_{k,2}\right) = \frac{N_1\mu_1 + N_2\mu_2}{N_1 + N_2}, \tag{3.5}$$

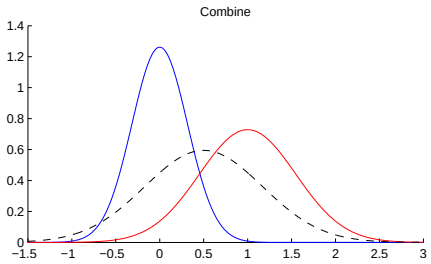
$$\sigma_{1,2}^2 = \frac{1}{N_1 + N_2}\left(\sum_{k=1}^{N_1} x_{k,1}^2 + \sum_{k=1}^{N_2} x_{k,2}^2\right) - \mu^2 = \frac{N_1(\sigma_1^2 + \mu_1^2) + N_2(\sigma_2^2 + \mu_2^2)}{N_1 + N_2} - \mu^2, \tag{3.6}$$

where we use the fact that

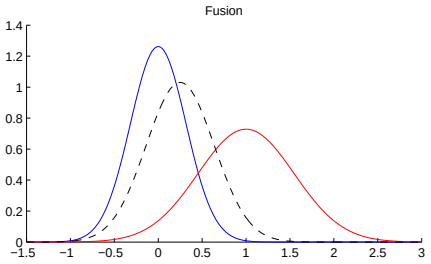
$$\sigma^2 = \frac{1}{N}\sum_{k=1}^{N}(x_k - \mu)^2 = \frac{1}{N}\sum_{k=1}^{N} x_k^2 - 2x_k\mu + \mu^2 = \frac{1}{N}\sum_{k=1}^{N} x_k^2 - \mu^2$$

$$\implies \frac{1}{N}\sum_{k=1}^{N} x_k^2 = \sigma^2 + \mu^2 \implies \sum_{k=1}^{N_1} x_k^2 = N(\sigma^2 + \mu^2) \tag{3.7}$$

This formulation corresponds with that of weighted mean and weighted variance, where N_1 and N_2 can be arbitrary weights or respective Gaussian distribution. The combining of two Gaussian distributions is illustrated in Figure 3.1a together with a comparison with the fusion of the two distributions.



(a) *Combining of two Gaussian distributions.*



(b) *Fusion of two Gaussian distributions.*

Figure 3.1: *Combining and Fusion of two Gaussian distributions with $N_1 = N_2 = 1$. The blue has mean 0 and variance 0.1. The red has mean 1 and variance 0.3.*

3.3 Combining and Fusion of Gaussian Processes

The combining of two Gaussian processes models is performed by the combination of equations 2.6-2.7 and 3.5-3.6 which results

$$\mu_{12}(x) = \frac{N_1\mu_1(x) + N_2\mu_2(x)}{N_1 + N_2}, \tag{3.8}$$

$$\sigma_{12}^2(x) = \frac{N_1(\sigma_1^2(x) + \mu_1^2(x)) + N_2(\sigma_2^2(x) + \mu_2^2(x))}{N_1 + N_2} - \mu_{12}^2(x), \tag{3.9}$$

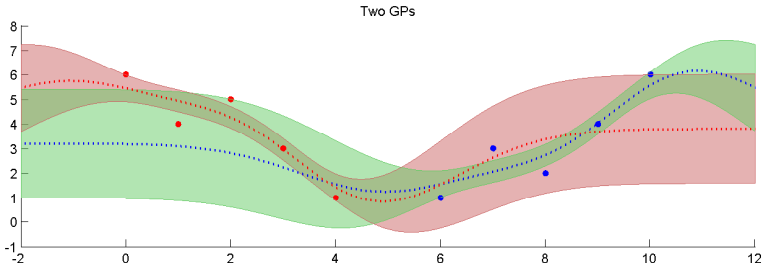
where N_1 and N_2 are weights of respective Gaussian process, similar to the weights in the previous section. For example, imagine that we now have observed Gaussian processes instead of Gaussian distributions. If the first GP model two observed GPs and the second GP only model one observed GP, then $N_1 = 2$ and $N_2 = 1$, and equation 3.8-3.9 would parametrize the combined GP of the three observed GPs. Fusion of two Gaussian process models with different evidence can be formulated similarly to equation 3.1-3.2 as

$$\sigma_{12}^2(x) = \frac{N_1 + N_2}{N_1\frac{1}{\sigma_1^2(x)} + E_2\frac{1}{\sigma_2^2(x)}}, \tag{3.10}$$

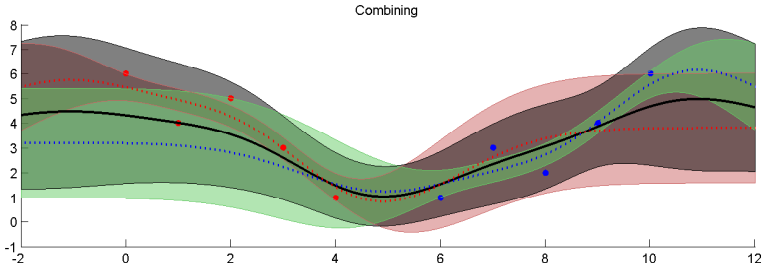
$$\mu_{12}(x) = \sigma_{12}^2(x)\left(\frac{N_1\frac{\mu_1(x)}{\sigma_1^2(x)} + N_2\frac{\mu_2(x)}{\sigma_2^2(x)}}{N_1 + N_2}\right), \tag{3.11}$$

where N_1 and N_2 are weights of respective Gaussian process.

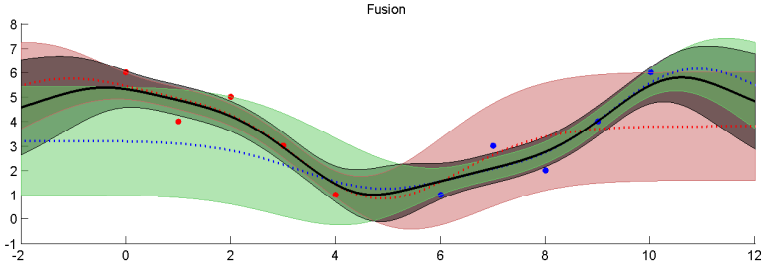
Fusing and combining two different local models that are serial, meaning that the intervals they primarily model are not overlapping, can be seen in Figure 3.2. Fusing and combining two different local models that are parallel, meaning that the intervals are overlapping, is shown in Figure 3.3. In the case of serial models we have several small models after each other which only describes a local part of the overall distribution that we want to model. What we want is for each local model to model the interval it describes the best and for the borders between models to be a smooth mix of the two bordering models, as would have been expected if it was a whole Gaussian process rather than a chain of small local models. In the case of parallel models we have several models that describe the same overlapping interval, and which can be seen as representing different estimates of the same common population which we want to model. We want a Gaussian process that combines the information from the different local overlapping models, in the same way that we combined Gaussian distributions in section 3.2. These figures show that fusion, which decreases uncertainty, is well suited for interpolation between serial local models. We also see that combining is well suited for merging parallel local models.



(a) *Two Gaussian processes. Support points of the red GP is red and support points of the green is blue.*

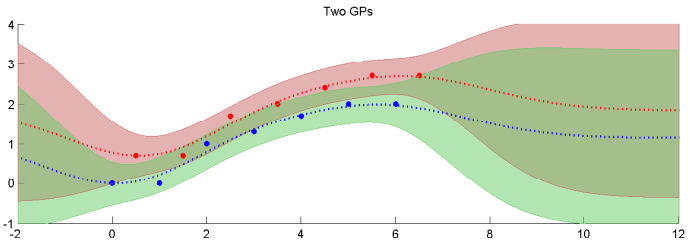


(b) *The combined GP is in dark gray and its mean is in black.*

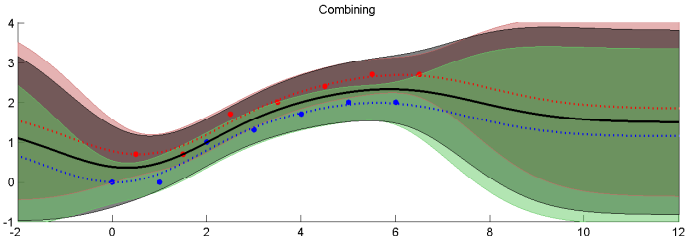


(c) *The GP fusion is in dark gray and its mean is in black.*

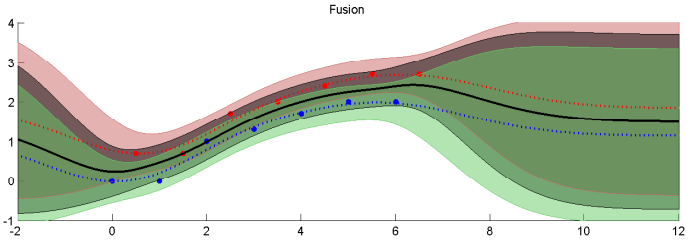
Figure 3.2: *Combining and fusion of two serial Gaussian processes. In both cases are the GPs weighted equally ($N_1 = N_2$).*



(a) *Two Gaussian processes. Support points of the red GP is red and support points of the green is blue.*



(b) *The combined GP is in dark gray and its mean is in black.*



(c) *The GP fusion is in dark gray and its mean is in black.*

Figure 3.3: *Combining and fusion of two parallel Gaussian processes. In both cases are the GPs weighted equally ($N_1 = N_2$).*

Changes to current sparse Gaussian process algorithms are however required in order for them to utilize the previously described combining of Gaussian processes. To demonstrate the applicability we have constructed a sparse Gaussian process algorithm **SGPUS** (Sparse Gaussian Process by Uniform Sampling) which uses an already existing or combined Gaussian process. It works by sampling the mean and variance functions of the target Gaussian process model with a uniform sampling interval, and can therefore at least reconstruct a Gaussian process under most circumstances as long as the Nyquist-Shannon sampling theorem (Nyquist, 1928) is fulfilled. This theorem states that a function g can be fully reconstructed from samples if g contain no frequencies higher than f hz and the sample frequency is two times that frequency.

3.4 Sparse Gaussian Process by Uniform Sampling

Sparse Gaussian Process by Uniform Sampling (**SGPUS**) is an algorithm that estimates a new GP from another GP (the target) by uniformly sampling the latter at the points that the new GP will use as support points. This means that we do not need to know the number of support points or kernel parameters of the target GP as long as we can evaluate the mean and variance at a number of points. The new GP will be as sparse as we specify it to be, but it will not capture the structure of the target GP if we make the new GP too sparse (i.e. too few support points). **SGPUS** approximates the target GP by making the new GP have the same mean value as the target GP at its support points, and the same variance as the target GP at and in the middle between each of its support points.

Let $\mathcal{GP}_{true}$ be the Gaussian process model that we want to model with the sparse Gaussian process model $\mathcal{GP}_{estimate}$. We assume that both GPs have zero mean and a Gaussian kernel as covariance function. Let the number of support points in $\mathcal{GP}_{estimate}$ be N.

Sample the mean and variance functions of $\mathcal{GP}_{true}$ at states $S = \{s_1, \ldots, s_N\}$ to produce the sampled states $S_{sampled} = \{\{\mu_{s_1}, \sigma^2_{s_1}\}, \ldots, \{\mu_{s_N}, \sigma^2_{s_N}\}\}$, by using equation (2.6-2.7):

$$\mu_{s_n} = \mu_{true}(s_n) = \mu_{\mathbf{y}} + k(s_n, \mathbf{x})(k(\mathbf{x}, \mathbf{x}) + \sigma_n^2 \delta_{nj})^{-1}(\mathbf{y} - \mu_{\mathbf{y}}) \tag{3.12}$$

$$\sigma^2_{s_n} = \sigma^2_{true}(s_n) = k(s_n, s_n) - k(s_n, \mathbf{x})(k(\mathbf{x}, \mathbf{x}) + \sigma_n^2 \delta_{nj})^{-1} k(s_n, \mathbf{x})^T \tag{3.13}$$

The new Gaussian process model is then defined by

$$\mathcal{GP}_{estimate} = \{\mathbf{x}_{new}, \mathbf{y}_{new}, \theta_{new}, \sigma^2_{new}\}, \tag{3.14}$$

where $\mathbf{x}_{new}$, $\mathbf{y}_{new}$ and σ^2_{new} are three vectors with corresponding input variables, output variables and output noise, respectively. θ_{new} is the old θ_0 and the new θ_1 parameter for the kernel. The two following sub-sections explain the details of estimating $\mathbf{y}_{new}$, σ^2_{new} and θ_{new}, using the following two vectors:

$$\begin{aligned} \mathbf{x}_{new} &= \mathbf{x}^* = \{s_1, \ldots, s_N\} \\ \mathbf{y}^* &= \{\mu_{s_1}, \ldots, \mu_{s_N}\} \end{aligned} \tag{3.15}$$

The final sub-section culminates in a formalization of the final algorithm **SGPUS**, which incorporates these estimations.

3.4.1 Variance Estimation

The variance $\sigma^2(x^*)$ of a Gaussian process for any point x^* is determined by the relation between x^* and $\mathbf{x}$ given by the kernel $k(x^*, \mathbf{x})$ as well as by the error σ_i^2 of respective $y_i \in \mathbf{y}$. In the case of the Gaussian kernel, the relation is governed by the exponential negative distance between x and x_i, weighted by a factor θ_1. Since the value of each x_i is assumed fixed, the free variables that can be changed are θ_1 and σ_i^2 for all $i = 1 \ldots N$.

By only optimizing over σ_i^2, the variance at each $x^* = x_i$ will be correct, but the

variance in between points in $\mathbf{x}$ is only correct in a special case. The special case is when the trajectory sampled is uniformly distributed in the support points $\mathbf{x}_{other}$ with a length between each successive x_i which is equal to the sample step length. The behavior of the variance between successive x_i is directly related to θ_1. In a sense σ_i^2 sets the minimal variance at each support point (x_i, y_i) and θ_1 describes how these are influencing the intermediate space.

The optimization over θ_1 and σ_i^2 is performed using gradient descent. Gradient descent is a first-order optimization algorithm that finds a local minimum by incremental steps in the direction of the negative of the gradient of an error function,

$$\phi_{n+1} = \phi_n - \gamma_n \nabla \mathcal{E}(\phi_n), \tag{3.16}$$

where ϕ_n is the parameter optimized at iteration n, $\mathcal{E}(\phi)$ is an error function parametrized by ϕ and γ_n is the step length which has to be small enough for convergence to be guaranteed. An initial guess ϕ_0 is assumed and the step size γ_n can be adjusted at each iteration to improve the convergence rate.

We want to minimize the difference in variance at the points $x_1 ... x_M \in \mathbf{x}^*$, at and in-between each support point, and therefore formulate the error function $\mathcal{E}$ as the least squares of this error:

$$\mathcal{E}(\theta_1, \sigma_1^2, \ldots, \sigma_N^2) = \mathcal{E}(\phi) = \frac{1}{2M} \sum_{k=1}^{M} (\sigma_{est}^2(x_k|\phi) - \sigma_{true}^2(x_k))^2 \tag{3.17}$$

The incremental update of gradient descent for each parameter we optimized over is

$$\phi_j = \phi_j - \gamma \frac{d}{d\phi_j} \mathcal{E}(\phi) \tag{3.18}$$

The parameter derivatives are derived in Appendix A and can with the substitutions $V = \left[k(\mathbf{x}, \mathbf{x}) + \sigma_i^2 \delta_{i,j}\right]$ and $e(\phi) = (\sigma_{est}^2(x_k|\phi) - \sigma_{true}^2(x_k))$ be formulated as

$$\frac{d}{d\theta_1} \mathcal{E}(\phi) = \frac{-1}{M} \sum_{k=1}^{M} \Big(2 \frac{dk(x_k, \mathbf{x})}{d\theta_1} V^{-1} k(x_k, \mathbf{x})^T + k(x_k, \mathbf{x}) \frac{dV^{-1}}{d\theta_1} k(x_k, \mathbf{x})^T \Big) e(\phi) \tag{3.19}$$

$$\frac{d}{d\sigma_i^2} \mathcal{E}(\phi) = \frac{-1}{M} \sum_{k=1}^{M} \Big(k(x_k, \mathbf{x}) V^{-1} \hat{e}_i \hat{e}_i^T V^{-1} k(x_k, \mathbf{x})^T \Big) e(\phi), \qquad i = 1 \ldots N \tag{3.20}$$

where $\hat{e}_i$ is the identity vector with zeros at all positions except for the $i-th$ place where there is a 1. $\hat{e}_i \hat{e}_i^T$ is therefore an N times N matrix filled with zeros except at the diagonal element (i, i) where there is a 1.

3.4.2 Mean Estimation

Updating the observation error affects the mean of the posterior such that it drifts. This is because the observation error σ_i^2 is optimized such that a matching variance is found as described in the previous sub-section. Since the observation

error affects how reliable each data point is in the Gaussian process model, we must now compensate the positioning of the output variables $\mathbf{y}^*$ so that $\mathcal{GP}_{estimate}$ evaluates to $\mathbf{y}^*$ for all variables in $\mathbf{x}^*$.

What we want is for equation 2.6 to equal the vector $\mathbf{y}^*$ for all $\mathbf{x}^*$, when we have the output variables $\mathbf{y}^*_{new}$ (which we want to find) and the input variables $\mathbf{x}^*$ as support points. We can formulate this using equation 2.6 if we assume $\mathbf{y}^*_{new}$ have zero mean, and we therefore use $\mathbf{y}^* - \mu_{\mathbf{y}^*}$ since this expression has zero mean as well. We want $\mathbf{y}_{new}$ to have the same mean as $\mathbf{y}^*$, so we can simply add this at the end.

$$\mathbf{y}^* - \mu_{\mathbf{y}^*} = 0 + k(\mathbf{x}^*, \mathbf{x}^*)\Big[k(\mathbf{x}^*, \mathbf{x}^*) + \sigma^2_{new,i}\delta_{ij}\Big]^{-1}(\mathbf{y}^*_{new} + 0) \tag{3.21}$$

The matrix $k(\mathbf{x}^*, \mathbf{x}^*)$ is of full rank and therefore invertible, this allow us to rewrite the formula as follows

$$\begin{aligned}
&\mathbf{y}^* - \mu_{\mathbf{y}^*} = 0 + k(\mathbf{x}^*, \mathbf{x}^*)\Big[k(\mathbf{x}^*, \mathbf{x}^*) + \sigma^2_{new,i}\delta_{ij}\Big]^{-1}(\mathbf{y}^*_{new} + 0) \\
&\implies \Big[k(\mathbf{x}^*, \mathbf{x}^*) + \sigma^2_{new,i}\delta_{ij}\Big]\Big[k(\mathbf{x}^*, \mathbf{x}^*)\Big]^{-1}(\mathbf{y}^* - \mu_{\mathbf{y}^*}) = \mathbf{y}^*_{new} \\
&\implies \mathbf{y}^*_{new} = \Big[k(\mathbf{x}^*, \mathbf{x}^*) + \sigma^2_{new}\Big]\Big[k(\mathbf{x}^*, \mathbf{x}^*)\Big]^{-1}(\mathbf{y}^* - \mu_{\mathbf{y}^*}) \\
&\implies \mathbf{y}^*_{new} = \sigma^2_{new}\Big[k(\mathbf{x}^*, \mathbf{x}^*)\Big]^{-1}(\mathbf{y}^* - \mu_{\mathbf{y}^*}) + (\mathbf{y}^* - \mu_{\mathbf{y}^*})
\end{aligned} \tag{3.22}$$

We introduce a regularization parameter λ to improve the numerical stability, as well as to stop too high resulting $\mathbf{y}$-values at the cost of allowing for some small estimation errors.

$$\mathbf{x}^*_{new} = \sigma^2_{new}\Big[k(\mathbf{x}^*, \mathbf{x}^*) + \lambda I\Big]^{-1}(\mathbf{x}^* - \mu_{\mathbf{x}^*}) + (\mathbf{x}^* - \mu_{\mathbf{x}^*}) \tag{3.23}$$

Finally we add the mean value of $\mathbf{x}^*$ which provides us with a closed form solution that corrects the values of $\mathbf{x}^*$ to the best possible given our regularization parameter,

$$\mathbf{x}_{new} = \mathbf{x}^*_{new} + \mu_{\mathbf{x}^*}, \tag{3.24}$$

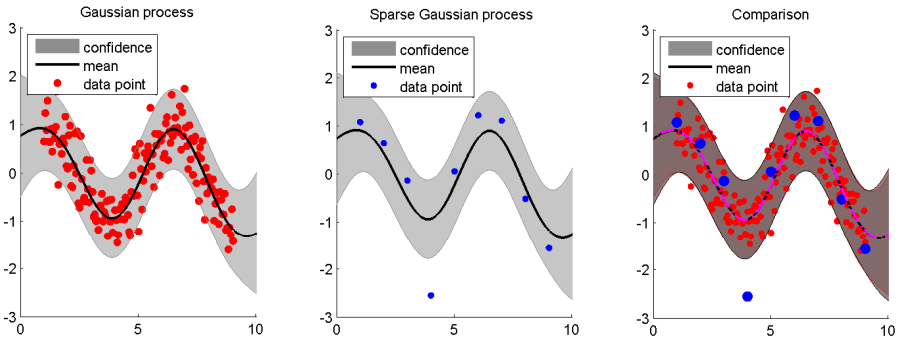
and the support-points' function value required for the Gaussian process model to evaluate its mean function to $y_i \in \mathbf{x}^*$ for $x_i \in \mathbf{x}^*$ for all $i = 1 \ldots N$ is thereby given by

$$\mathbf{x}_{new} = \sigma^2_{new}\Big[k(\mathbf{x}^*, \mathbf{x}^*) + \lambda I\Big]^{-1}(\mathbf{x}^* - \mu_{\mathbf{x}^*}) + \mathbf{x}^* \tag{3.25}$$

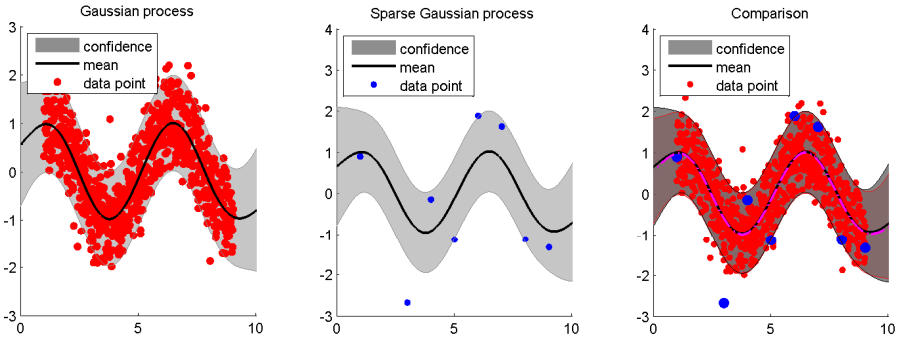
3.4.3 The SGPUS Algorithm

The details of the **SGPUS** are shown in Algorithm 1. The input $s_{1\ldots N}$ is a vector of uniformly distributed real numbers and corresponds to the input variables in the support points. $\mu_{true}(s_{1\ldots N})$, $\sigma^2_{true}(s_{1\ldots N})$ can be evaluations of any mean and variance function $(\mu(x), \sigma^2(x))$ for any argument $x \in \mathbb{R}$, for example the combining or fusion of two Gaussian processes described in sub-section 3.3. In the case that it is another Gaussian process it would be formulated like this:

$$\mu_{true}(s_{1\ldots N}), \sigma^2_{true}(s_{1\ldots N}) \leftarrow \text{Evaluate}\mathcal{GP}_{true}(s_{1\ldots N})$$



(a) *161 support points.* ***(b)*** *9 support points.* ***(c)*** *Comparison*



(d) *801 support points.* ***(e)*** *9 support points.* ***(f)*** *Comparison.*

Figure 3.1: *SGPUS applied to a Gaussian process optimized over 161 (top) respectively 801 (bottom) data points generated from a sine wave with $\mathcal{N}(0, 0.5^2)$-distributed noise. In the comparison the true mean is magenta colored and the true confidence is colored red. A close to perfect overlap is demonstrated within the range of the data. The optimization of the Gaussian process took 0.48s for the top and 10.9s for the bottom. SGPUS took 0.59s for the top and 0.78s for the bottom.*

Two example of runs of the **SGPUS** algorithm are shown in Figure 3.1, where a GP model is first found for a set of observed points and which then is made sparse. The sparse GP and the original GP are compared and they deviate only outside of the range of the data. That they deviate can be seen as an issue for certain applications, but it is of marginal concern in our application as well as in other applications where only the GP model within the range of the data is relevant.

"Evaluate $\mathcal{GP}_{estimate}(s_{1...N} \mid \phi, \mathbf{y})$" in Algorithm 1 refer to the evaluation of Equation 3.12-3.13 using $\mathbf{y}$ and the parameters in ϕ which provide the support point variance (observation noise) and the kernel hyper parameter θ_1.

Algorithm 1 Sparse Gaussian Process by Uniform Sampling

1: **procedure** SGPUS($s_{1...N}$, ϕ_{init}, $\mu(x)$, $\sigma^2(x)$)
2: $\phi \leftarrow \phi_{init}$
3: $\mu_{true}(s_{1...N}) \leftarrow \mu(s_{1...N})$
4: $\sigma^2_{true}(s_{1...N}) \leftarrow \sigma^2(s_{1...N})$
5: $\mathbf{x} = \mu_{true}(s_{1...N})$
6: $\mu_{est}(s_{1...N}), \sigma^2_{est}(s_{1...N}) \leftarrow$ Evaluate $\mathcal{GP}_{estimate}(s_{1...N} \mid \phi,\ \mathbf{x})$
7: **for** i = k..**maxIterations do**
8: $\theta_1 \leftarrow \theta_1 - \gamma_1 \frac{d}{d\theta_1}\mathcal{E}(\phi \mid \sigma^2_{est}(s_n), \sigma^2_{true}(s_n))$
9: $\sigma_i^2 \leftarrow \sigma_i^2 - \gamma_2 \frac{d}{d\sigma_i^2}\mathcal{E}(\phi \mid \sigma^2_{est}(s_n), \sigma^2_{true}(s_n))$
10: $\phi \leftarrow \{\theta_1, \sigma_1^2 \dots \sigma_N^2\}$
11: $\mu_{est}(s_{1...N}), \sigma^2_{est}(s_{1...N}) \leftarrow$ Evaluate $\mathcal{GP}_{estimate}(s_{1...N} \mid \phi,\ \mathbf{x})$
12: **if** $\|\nabla\mathcal{E}(\phi \mid \sigma^2_{est}(s_n), \sigma^2_{true}(s_n))\|_2 \leq stopCriterion$ **then**
13: **break**
14: Adapt γ_1, γ_2
15: $\sigma^2_{new} \leftarrow \sigma^2_{1..N}$
16: $\theta_{new} \leftarrow \theta_1$
17: $\mathbf{x}^* \leftarrow \mu_{est}$
18: $\mathbf{y}_{new} \leftarrow \sigma^2_{new}\left[k(\mathbf{x}^*, \mathbf{x}^*) + \lambda I\right]^{-1}(\mathbf{y}^* - \mu_{\mathbf{y}^*}) + \mathbf{y}^*$
19: **return** $\mathbf{y}_{new}$, θ_{new}, σ^2_{new}

The complexity of Algorithm 1 is $\mathcal{O}(N^3)$ plus the complexity of line 3 – 4 in the number of support points N. For all intended use-cases in this thesis it requires the evaluation of at least one Gaussian process to produce $\mu_{true}(s_{1...N})$ and $\sigma^2_{true}(s_{1...N})$, meaning that the mean and variance function ($\mu(x)$ and $\sigma^2(x)$) taken as arguments correspond to equation 2.6-2.7 or weighted combinations of several such equations. As a consequence the total complexity of the application of **SGPUS** is $\mathcal{O}(NM^2 + N^3)$ where M is the maximum number of support points among the Gaussian processes used. It is multiplied with N due to the N evaluations necessary to compute N means and variances for the vectors $\mu_{true}(s_{1...N})$ and $\sigma^2_{true}(s_{1...N})$.

The adaptation of γ_1 and γ_2 currently performed is very trivial. We don't adapt γ_2 at all, and γ_1 is divided by half when the direction of the gradient on line 9 changes and multiplied by two when it does not change. A more sophisticated line-search of both variables can probably improve the convergence time.

The kernel parameter θ_0 is currently not optimized, and this might be a drawback. We cope by taking the highest θ_0 of all the GPs used, but this is sidestepping the algorithm and not a neat solution.

A possible extension is to support non-uniform sampling, which would require multiple smoothness parameters such as in equation 2.11. This would probably be slower due to additional variables to optimize over, but it would also make it

possible to have more support points at changing parts of the GP and less points at constant or slowly changing parts. Since Gaussian processes are infinitely differentiable we could spend more time on putting support points at the minima and maxima (as well as on suitable, rapidly changing places) of the observed GP, which would improve the modeling accuracy and require fewer support points in total. We would as a consequence no longer be bound by the sampling theorem (Nyquist, 1928) for assurances of being able to capture the structure of the observed GP.

4

Activities

This chapter first provides an overview of what we mean with an activity within the scope of this thesis, and provides an intuition to how to learn activities and their usefulness. After the first two introductory sections the more formal activity model is introduced. The chapter concludes with how the number of observed trajectories supporting an activity is used to make a fair comparison between activities when they compete for explaining an observation and when comparing how similar two activities are. Activity similarity and activity classification is introduced briefly.

4.1 Activities

In the context of this thesis, an activity can informally be defined as something a particular object does. In our case, an instance of an activity 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 a slow-down where the velocity state variable is expected to decrease. An activity model should capture such continuous developments.

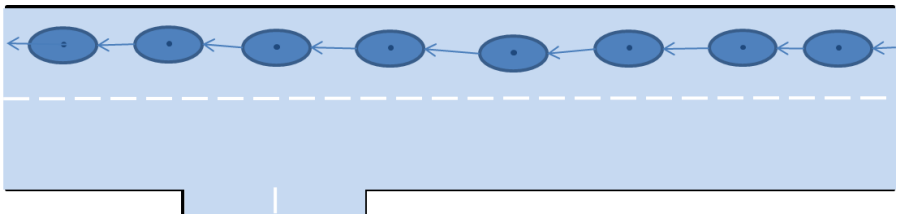
When considering the context in which an activity is performed, it is natural to consider in which ways either the context or the activity is likely to change. Change is intrinsically causal and we can consider both prediction and explanation as exemplified by the following two questions: Given that a car is entering a T-crossing by its main road and is slowing down, how likely is it that the car will turn rather than drive straight ahead? Given that a car leaves a T-crossing, how likely is it that it did turn? Every dynamic object is perceived to traverse an activity chain, by transitioning from one activity to another.

The type of activities of primary concern in this thesis are so called spatio-temporal activities. These are activities that are dynamic in the sense that they vary spatially in the state space over time. Such activities are for example a slow-down, cruising at constant speed or turning. Activities that do not fall into this category are activities such as standing still, where there is no spatial change over time. Inclusion of stationary activities are left for future work.

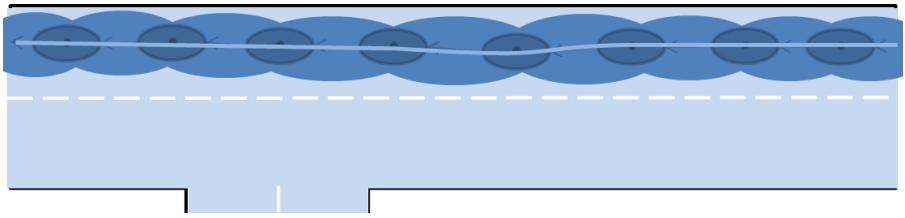
4.2 Modeling Spatio-Temporal Activities

The idea of modeling spatio-temporal activities is as follows. Use observations to model the behavior of variables through space and time. Segment the world based on where multiple models occur. If a new observation is poorly represented by the current model, create a new local model. If multiple models are similar, merge these.

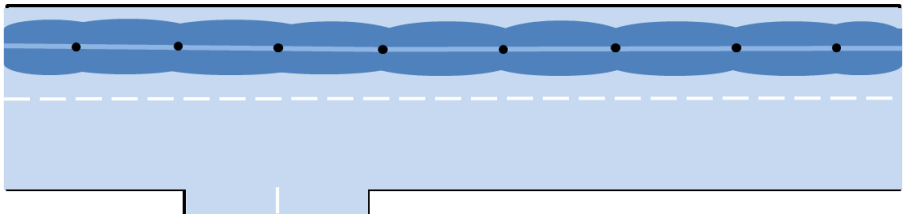
Consider that we are observing the positions of a car in motion on a road, a scenario shown in Figure 4.1. We assume that we can measure the position as a 2D Gaussian distribution, with a mean and a covariance matrix which provide the most likely position and the uncertainty in 2 dimensions. Instead of treating each position separately, we can apply the prior knowledge that each position is a part of a trajectory and we as observers know the ordering of the observed positions (Figure 4.1a). The continuous trajectory of discrete positions can be modeled using a Gaussian process (Figure 4.1b), which is a probability distribution over functions that in this example are possible 2D-trajectories. The uncertainties in each of the observed 2D positions are incorporated in the Gaussian process. Any given point on the Gaussian process trajectory model we now have is a Gaussian distribution, and we can easily generate new random trajectories which follow the characteristics of the observed trajectory.



(a) A trajectory of individual measurements of the 2D position of a vehicle with Gaussian distributed measurement noise.



(b) A Gaussian process modelling the 2D trajectory.



(c) A more confident Gaussian process model, by incorporating multiple observed trajectories in the same lane.

Figure 4.1: *From observed discrete 2D positions to a continuous trajectory.*

By observing more cars and their ordered positions, the trajectory model can be updated by combining the observed trajectory and the previous trajectory model and applying uniform re-sampling as described in Chapter 3. This allows the model to incorporate more observations of trajectories while keeping the model complexity constant. The model complexity for the Gaussian process is the number of support points, shown as black dots in the figure. If the trajectories of the observed cars are somewhat similar, an even more certain Gaussian process should emerge as a well-supported trajectory model for the observed road (Figure 4.1c).

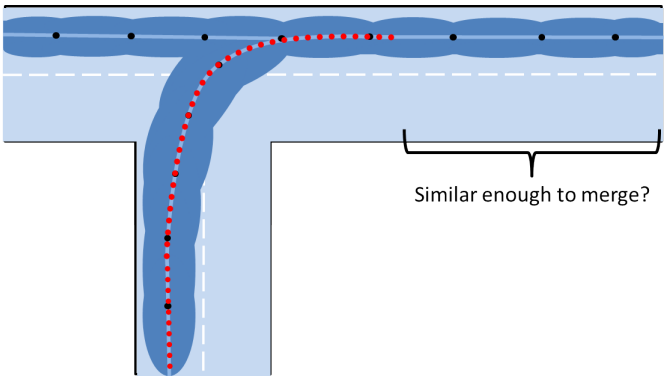


Figure 4.2: *The trajectory of a vehicle turning in the crossing is also modeled using a Gaussian process.*

An observed car might turn in the T-crossing, not following the old trajectory model (Figure 4.2). Along the turning trajectory before the turn, there will be a 2D position where the turning car is outside of the confidence interval of the straight going trajectory model. The position of the car has up till this point been explainable by the straight model, and can possibly be used to update the straight trajectory model. After this point the car is observed to perform an unexpected and previously unknown behavior. It is possible to discriminate between the old behavior and the new behavior from this point on. The old straight trajectory model is therefore split in to two at the point and a new turning trajectory model is created which originates from that point (Figure 4.3).

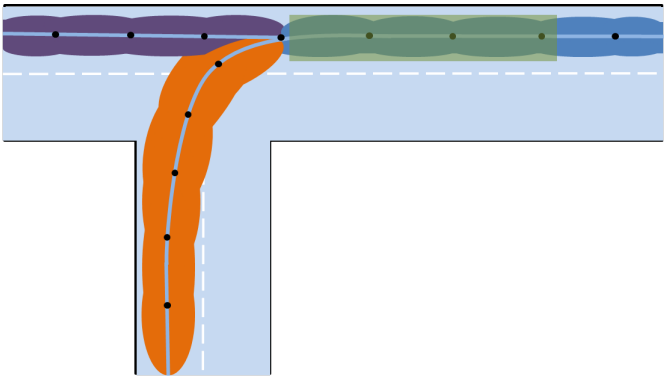


Figure 4.3: *The previously learned trajectory is segmented into two shorter trajectories, where one corresponds to the overlap with the observed trajectory. The point of segmentation is where the observed trajectory stopped overlapping sufficiently with the previously learned activity. The marked area shows where cars that intend to turn left are assumed to be slowing down as the turn approaches.*

By modeling more than just the trajectory of 2D positions, such as the 2D velocity

as well, it will be possible to detect and predict the turn even earlier than the current branching point. Cars usually drive past in higher speeds than what is practical to turn in, so those that turn have to slow down before turning. Even though we are only observing positions we can observe the velocity via a virtual sensor by using a Kalman filter. With both the position and velocity trajectory modeled, the activity of driving straight and that of turning left are dissimilar earlier than the previous branching point due to the variations in velocity (Figure 4.4).

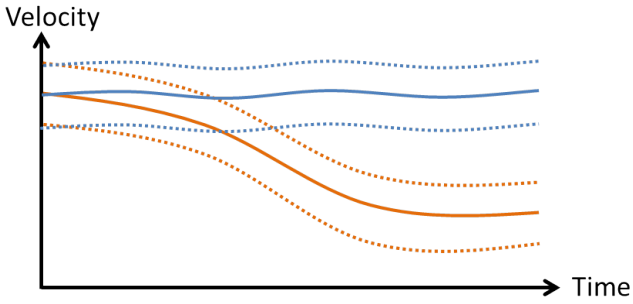


Figure 4.4: *The two different trajectories the speed is expected to follow for cars driving in the previously marked area. The blue trajectory models the vehicles that drive straight ahead and the red trajectory models those that turn left. We assume that no vehicle ever stops or waits because vehicles in front of it is slowing down or stopping e.g. to perform the left turn. Handling this is left for future work.*

4.3 The Activity Model

An activity is 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. To model a path through the state space over time we use a Gaussian Process which provides a continuous non-linear mapping with confidence measures using a finite number of data points here called support points. 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 mapping is modeled using a Gaussian process model $y = f_{GP}(t)$ and it is used to classify new observations and to support prediction, similar to Kim et al. (2011). The assumption is that an activity is a curve in an N-dimensional state space. The activity model is the result of the continued accumulation of explained observed trajectories. The amount of these supporting trajectories is seen as the amount of evidence E of the activity model, and is used in comparison with other activity models.

Given an observed state, the parametrization variable t of the observed trajectory must be known in order to calculate how well an activity explains the observation.

In Kim et al. (2011) 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. Further, they only consider entire trajectories without temporal segmentation, and with a state space consisting of positions and velocities. 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, $t = g_{GP}(y) = f_{GP}^{-1}(y)$, in order to estimate the mapping from state space to the temporal parameter. For the 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. The inverse mapping is constructed by switching the input and the output, $\mathbf{x}$ and $\mathbf{y}$, of f_{GP}.

The activity model $\mathcal{M}$ (Definition 4.1) consists of two Gaussian process models, modeling the parametrized state trajectory and its inverse mapping. These two models are in turn both composed of a set of finite input-output pairs together with hyper parameters for the covariance function. The covariance function used in both cases is the squared exponential kernel (Equation 2.8), which is commonly used in Gaussian process regression.

4.1 Definition (Activity Model). An activity model $\mathcal{M}$ is formally defined as a pair of Gaussian process models f_{GP} and g_{GP}. f_{GP} is a mapping from the time parameter $t \in [0.0\ 1.0]$ to the D-dimensional state space $y \in \mathbb{R}^D$, with kernel parameters θ_f. g_{GP} is the inverse mapping of f_{GP}, with kernel parameters θ_g.

$$\mathcal{M} = \langle f_{GP},\ g_{GP} \rangle \tag{4.1}$$

$$f_{GP} = \langle t,\ y,\ \theta_f \rangle \tag{4.2}$$

$$g_{GP} = \langle y,\ t,\ \theta_g \rangle \tag{4.3}$$

The main purpose of the inverse mapping is to project a state space point $x^* \sim \mathcal{N}(\mu_{x^*}, \sigma_{x^*}^2)$ 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 point on one trajectory being explained by a point on another. The calculation is performed by approximating x^* as μ_{x^*} and t^* as μ_{t^*},

$$y^* = f_{GP}(\mu_{t^*}), \qquad t^* = g_{GP}(\mu_{x^*}). \tag{4.4}$$

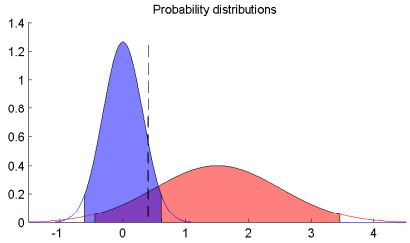
The latter approximation is valid if the inverse mapping accurately models the inverse mean function. The former approximation can be improved by techniques for uncertain inputs to GPs described in (Girard et al., 2002).

4.4 Evidence

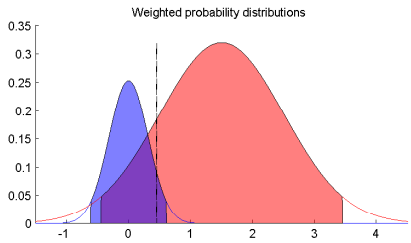
Evidence is used in this thesis as a measurement for how much support an activity has had from the observations in the past. The amount of evidence for a given activity is the number of observed trajectories this activity has explained so far.

The mean of the Gaussian process is used to model the expected trajectory of the activity model. The variance of the Gaussian process is used to model both the uncertainty of the activity model (in state space proximity) and the expected wideness of the activity model in the state space. For example, the 95% confidence interval might be wide enough to contain all possible trajectories on a single lane, and thereby capturing the activity in this region. On the other hand, the observations can be very uncertain and sporadic which makes the 95% confidence wide as well. This introduces a problem of ambiguity of the wideness of a trajectory, between uncertainty and modeled structure, and it is solved by relating the amount of evidence of activities.

The classification of observed trajectories is presented in the next section and is based on local conditional probabilities similar to those used in (Kim et al., 2011). This makes an activity which has explained a lot of observations in the past, but is wide, less likely to explain an observed trajectory than a less supported more narrow activity in some cases (Figure 4.5a).



(a) The slices of two Gaussian processes are show as two Gaussian distributions. Before evidence weighting.



(b) The slices of two Gaussian processes are show as two Gaussian distributions. After evidence weighting.

Figure 4.5: *Two Gaussian distributions with different evidence, and an observed point indicated as a black dashed line. The blue has mean 0, variance 0.1 and evidence 1. The red has mean 1.5, variance 1.0 and evidence 4.*

This ambiguous duality is handled by having a weighting evidence measure for each activity, based on the number of observations that support each activity. An activity supported by a large amount of observations is likely to have a shape that starts to capture the underlying structure of the observed fluents composing the activity. The shape should eventually start to converge to a stable form as sufficient observations reinforce it. The weighting of the evidence amount is a way to make the well-supported activity more resistant to change and new observations.

If an activity with immense support from observations has a large variance, then it is due to the shape of the activity.

For example, consider the case showed in Figure 4.5 where a comparatively strongly supported activity (red) is much wider than the less strongly supported activity (blue). An example scenario can be that what is shown is a slice of a road lane right before a road exit, where the blue Gaussian distribution models the position of the cars that leave the main road for the exit and the red Gaussian distribution models the position of cars that drive past the exit. The cars that will use the exit tend to position themselves close to the road lane's border, while the cars passing the exit tend to position themselves more uniformity around the road lane's center although also at the road lane's boarder from time to time. If most cars pass by the road exit then it is more likely that a car close to the lane border will drive past the exit compared to if cars more often exit the road than continue past the exit. However, since the red activity model is wide enough to cover the entire lane it is without any adjustments less likely to explain cars at the lane border compared to the blue activity, regardless if driving on the border are a hundred times more likely to continue than to exit the road. By weighting the activity likelihoods with the number of explained observations, this problem is approximately reduced.

4.5 Activity similarity

Activities, as well as observed trajectories and activities, are compared based on a similarity measure using point-wise conditional probabilities over the range of both of them. When considering an observed trajectory, the activity which has the highest local likelihood for a point on the observed trajectory is regarded as the activity that best explains that local point of the observed trajectory.

Let A be an activity consisting of $\langle f_{GP}, g_{GP} \rangle$ and let $x^* \sim \mathcal{N}(\mu_{x^*}, \sigma^2_{x^*})$ be an observed point or a point on another activity. If the different dimensions of the state space are assumed to be independent, the *local likelihood* (Kim et al., 2011) of x^* given A is

$$P(x^*|A) = P(x^*|y^*) = \prod_{d=0}^{D} P(x^*_d|y^*_d), \qquad y^* = f_{GP}(g_{GP}(x^*)), \tag{4.5}$$

where D is the dimension of the state space and y^* is a D-dimensional multivariate Gaussian distribution. When considering multiple activities the calculation of the local likelihood is weighted by the relative evidence E in support of respective activity. If T_n is point n on trajectory T then

$$P^*(T_n|A_k) = \frac{E_k}{\sum_i (E_i)} P(T_n|A_k), \tag{4.6}$$

where i ranges over all activities in consideration of which A_k is one. This allows a well supported activity with high variance to keep significant importance. Activity similarity is for example used to determine what activity best explains a

newly observed trajectory.

5

The Activity Learning and Classification Framework

In this chapter we present a framework that is able to unsupervised learn connected atomic activity representations from observed trajectories, by discriminating between intervals that are and are not similar enough to previously-learned activities. The chapter starts with a top-down overview of the framework, followed by an in-depth description of the activity learning mechanisms. The chapter is concluded with a description of the recognition and prediction capabilities of the framework.

5.1 The Framework

The proposed framework (Figure 5.1) learns 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. The framework can also be used to predict how abnormal the current trajectory is and how likely future activities are.

The framework models activities as Gaussian Processes, which are sufficiently expressive to model the development of the state variables including uncertainties in observations. Activities are learned as edges in a graph, where the nodes are intersection points, in the state-space, between activities. On a higher abstraction level activities are seen as nodes, with edges representing transitions which are weighted according to the observed empirical probability of the transitions. The latter construct enables prediction of activities through probabilistic inference. The graphs are called the *State Space Graph* and the *Activity Transition Graph*.

An example is shown in Figure 5.2. The Activity Transition Graph is a causally

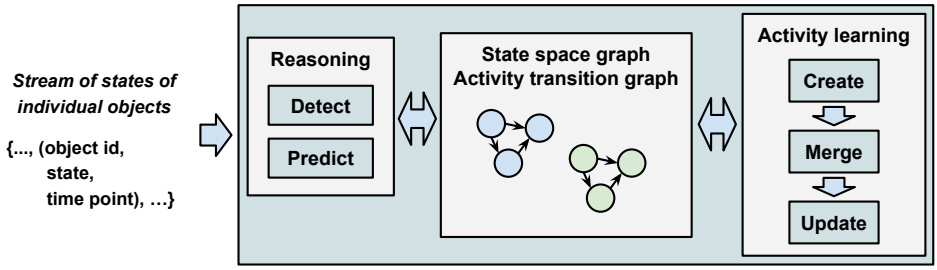
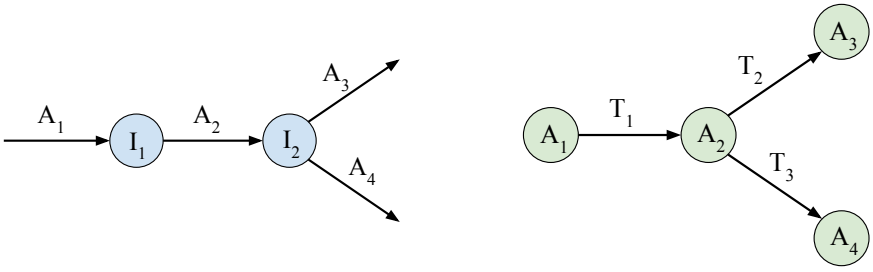


Figure 5.1: *An overview of the framework. It is built around the three modules* Reasoning, Activity Learning *and the knowledge base consisting of two graphs, which the two former modules make extensive use of. The* Activity learning *takes entire observed trajectories and updates the graphs. It is divided into three parts which are performed sequentially. The* Reasoning *module detects and predicts activities based on observations.*

ordered discrete Markov chain. We currently only consider 1-order Markovian transitions in the formulation of this graph. However, a natural future extension is to use a higher order Markov chain. This would allow the activities to affect each others likelihood over longer distances. This does however introduce some additional issues, such as how to keep the number of transitions sparse while finding out which transitions that are statistically significant and should be kept. Such an extension is left for future work.



(a) *State Space Graph* ***(b)*** *Activity Transition Graph*

Figure 5.2: *A_i denotes an activity in both the State Space Graph and the Activity Transition Graph, T_j denotes an intersection point in the state space, I_w 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.*

5.2 Activity Learning

To learn activities from observed trajectories, the framework segments trajectories into sequences of activities, where each activity differs from all other activities currently known. Activities are segmented such that no two activities can have overlapping intervals in the state space, where the margin for overlap is controlled by $s_{threshold}$. Figure 5.3 illustrates the creation of a new activity by trajectory segmentation, and a larger example of activity learning using the T-crossing domain is shown in Figure 1.1, page 4.

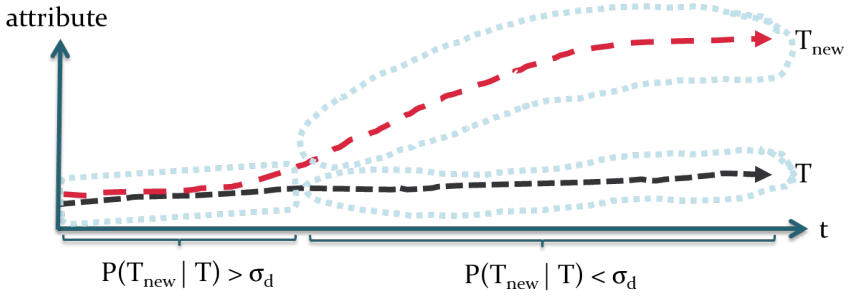
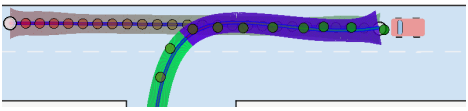


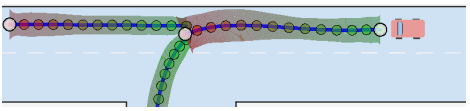
Figure 5.3: *Activity Learning. Both the old activity trajectory T and the new activity trajectory T_{new} will be split into two activities, where one part of each is overlapping and one is not. The overlapping parts will be merged, to combine the information of the two into a new activity, with added evidence from the multiple observations. The non-overlapping part of the new activity is now an activity branching out from the mean point of the split between overlap and non-overlap. This local part of the activity graphs now has an activity that leads to a branching into two possible other activities.*

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 from the observed trajectory, and then to generate a new trajectory by applying SGPUS to the GP model (chapter 3) to get a smaller number of support points for computational efficiency. The next step is to classify the trajectory given the current set of activity models. Based on this classification the graphs are updated. 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 are used to update that activity with the new observed information. See Figure 5.3 for an example. 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.

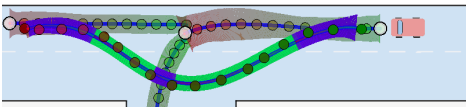
By letting the transition nodes store statistics of the transitions between the ac-



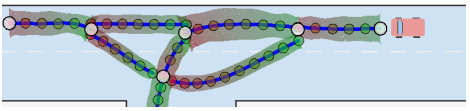
(a) *An activity of driving straight and an observation of a left turn. Bright blue indicates interval for update. Bright green indicates a interval which will result in the creation of a new activity.*



(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.*



(c) *An illegal overtake is observed. The rounded trajectory of the overtake is indicated to be created as two activities since it is too similar to the left turn when spatially crossing it. This does not happen if velocities are part of the state space.*



(d) *After the overtake trajectory has been learned.*

Figure 5.4: *Example of the learning of the State Space Graph using only 2D positions as state variables for the activities, in order (a)-(d). Activity coloring is from green (t = 0.0) to red (t = 1.0). The white circles are intersection points, the other circles are support points of the activities' f_{GP}.*

tivities, the Activity Transition Graph becomes another perspective of the same information contained in the State Space Graph. It is updated simultaneously with the State Space Graph in the different learning steps.

The inner workings of the framework when updating its knowledge base with a new observed trajectory is demonstrated in Figure 5.5. In Figure 5.5 (c) the different steps of the learning module are shown. 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 local likelihoods on the y-axis of the plot are 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 colored red and the non-ambiguous ones colored blue. The fifth and final graph shows the operations planned to be executed for each interval. After generating the set of planned operations such that the resulting knowledge base's structure is consistent, then the create, merge and

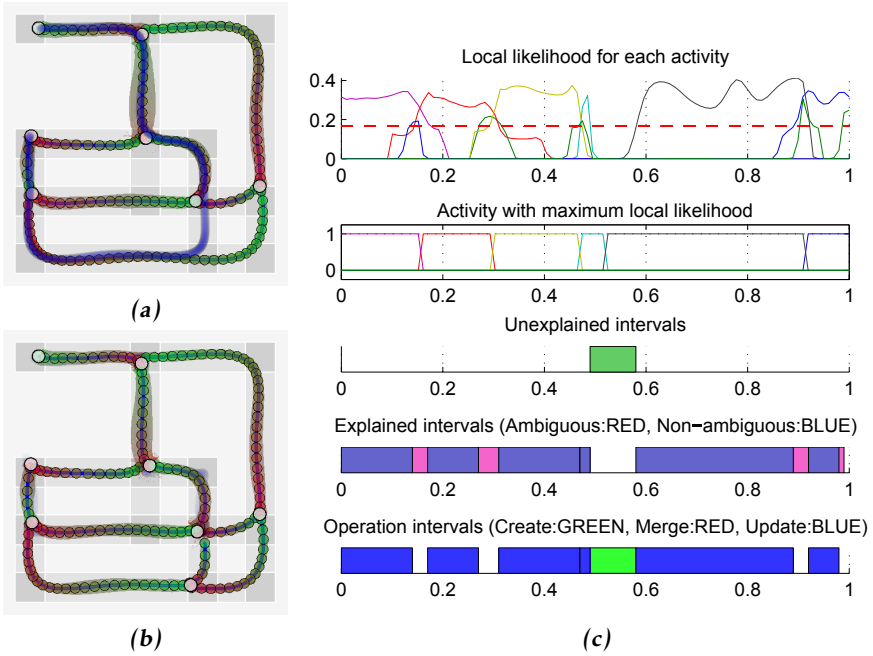


Figure 5.5: *The State Space Graph before* ***(a)*** *and after* ***(b)*** *learning a new observed trajectory (shown as a thick blue line). The observed trajectory starts at the top-left of the graph.* ***(c)*** *The different stages of analysis performed by the framework as described in the text.*

update operations are performed in that order. Whether the different operations are added to the plan or not depends on the rules for each operation and these are explained in the upcoming subsections.

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. 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 they should not be considered for merging.

The remainder of this chapter explains the operations of learning new activities (5.2.1), merging activities (5.2.2) and updating activities (5.2.3) as implemented in the framework. The activity learning section is concluded with Section 5.2.4 about separating learned activities. Here we elaborate on possible complementary extensions to these three operations that are supposed to improve both re-

producibility and completeness of the framework.

5.2.1 Learning New Activities

Given a observed trajectory T, a new activity is created for each maximal interval $\tau := [t_0 : t_1]$ where $P(T_\tau|A_k) < s_{threshold}$ for all activities A_k in consideration and where $\Delta l_\tau > l_{threshold}$. That is, any long enough interval of the observed trajectory, which is insufficiently explained by every concerned activity, is added to the framework as a new activity.

Intersection points are used to connect activities with other activities. Since activities are not allowed to be shorter than $l_{threshold}$, it is also the case that two intersection points cannot be created on the same activity with a distance between them shorter than $l_{threshold}$.

If needed, a new intersection point is created at each end of the new activity, but only if the length to the nearest other intersection point is larger or equal to $l_{threshold}$. If not, the new activity is connected to the nearest intersection point and the intersection point is updated to be placed at the weighted mean position of the end-points of the connected activities. The weighting is done in accordance to the amount of evidence of each connected activity. Figure 5.6 show both cases.

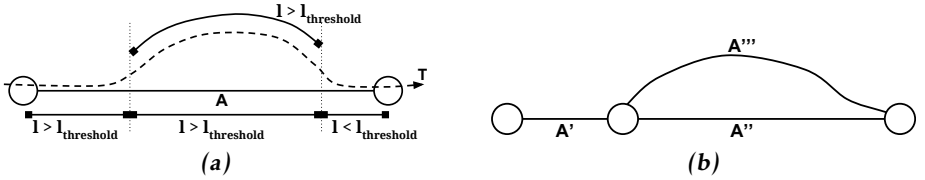


Figure 5.6: *Illustrative example of the* create procedure*, where the left figure is before and the right figure is after. The dashed lines indicate where the similarity of A and T is crossing the* $s_{threshold}$.

If a new activity is branching into or out of a previous activity, the previous activity is split in two at the branching point. The intersection point is placed at the weighted mean position. Intersection points that are closer than $l_{threshold}$ to each other are merged into a single intersection point. An illustrative example is shown in Figure 5.6, where a new observation is branching out of the known activity and a new intersection point is created due to the splitting.

5.2.2 Merging Activities

To remove redundant activities, activities are merged at intervals that are too similar to each other. Activities are also merged in order to allow for spatially wide activities, which might be initially observed as several parallel activities growing together over time as more observations are acquired.

All activities A_k are merged with A^* on the interval $\tau := [t_0 : t_1]$ for which $P(T_\tau|A_k) \geq s_{threshold}$, $\Delta l_\tau > l_{threshold}$ and A^* is the activity with the maximum

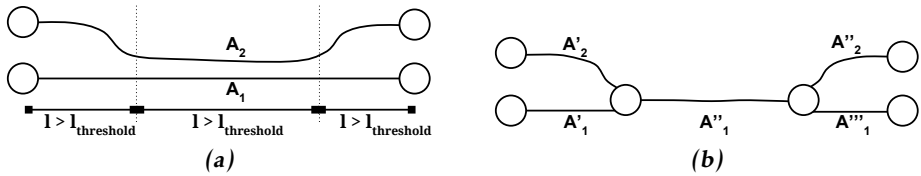


Figure 5.7: *An illustrative example of the* merge procedure*, where the left figure is before and the right figure is after. The dotted lines indicate where the similarity of both A_2 and A_1 are exceeding the $s_{threshold}$.*

similarity $P(T_\tau|A^*)$. That is, all activities which sufficiently explain long enough intervals of the observed trajectory are merged with the activity best explaining that interval.

An activity is merged into another by applying uniform re-sampling, as described in Chapter 3, of the combined activity models. The combining is weighted with the evidence of the respective activities. The final activity has the combined amount of evidence from all activities merged. Intersection points are created and updated in the same way as when new activities are created. An illustrative example is shown in Figure 5.7 where two activities get too close (become too similar) to each other and parts of each are merged.

5.2.3 Updating Activities

Any interval τ_i on T not used for creating or merging in the two previous steps carry information that is used to update activities. The activity best explaining such an interval cannot have been created or been part of a merge in the previous two steps.

The intervals I_n $(n = 1...N)$ remaining with $length > l_{threshold}$ always have at every point an activity A^* with maximum similarity where $P(T_t|A^*) \geq s_{threshold}$ holds except for intervals with $length \leq l_{threshold}$. These intervals are segmented into shorter intervals such that there is only a single activity with maximum similarity for the entire interval, $I_{n,i}$.

If $length(I_{n,i}) \leq l_{threshold}$ then the interval is combined with the neighbor of shortest length until they are composite intervals Ic satisfying $length(Ic_{n,i}) > l_{threshold}$. The activity assigned as the most likely to the composite interval is the activity A_k which has the highest $P(Ic_{n,i}|A_k)$ for all k in consideration.

The observed trajectory is modeled as an activity so that intervals or composite intervals of it can update the most similar activity by merging the activity with the intervals in question. This is a special case of the merging of two activities, with the weight of the observed intervals' activity equal to one since it only represents a single observation.

5.2.4 Separating Learned Activities

It is possible to construct scenarios where some activities grow to cover what would otherwise have become two different activities side by side. This could happen either due to bad luck of the ordering of the observed trajectories or due to new activities emerging after a while within the space of previous activities (Figure 5.8).

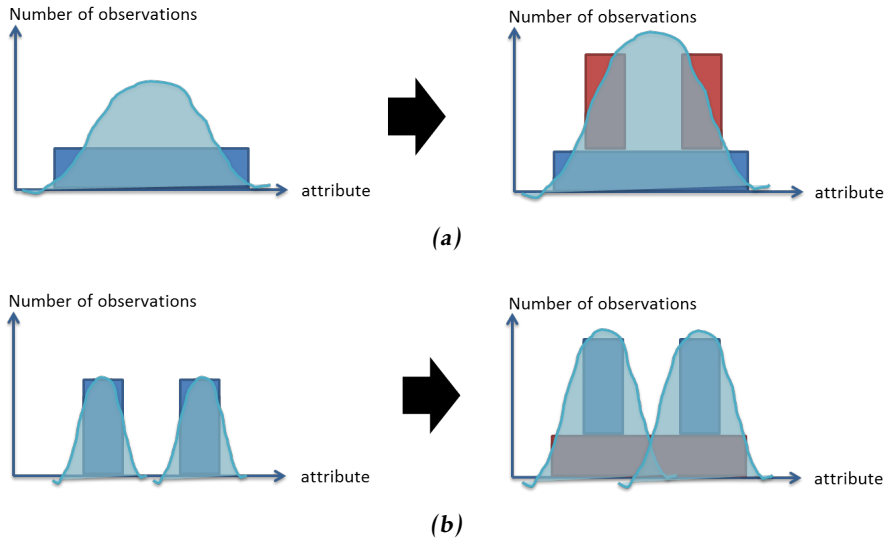


Figure 5.8: *An illustrative example of the how the order of observations in some cases can create a clustering problem. Each graph has the Gaussian distribution in transparent blue indicating how the activity has modeled the observed data. The blue box(es) indicate the distribution of the observations seen first and the red box(es) indicate the distribution of observations seen later. In (a) the problem occur while in (b) it does not, with different end result of activity learning and modeling*

The problem is a clustering problem, where a current activity no longer represent the data sufficiently and should be split in two (or more) activities. This is illustrated in Figure 5.9 where the trajectory T of an activity should be split into two activities with respective trajectory T_A and T_B.

This could potentially be solved using standard clustering algorithms such as Expectation Maximisation (Bishop, 2006) but there is a big risk that these would take a very long time to run since they would need all observed trajectories. Storing a history of all the observed trajectories for a system running the framework is also a potential problem since the purpose of the framework is to handle continuous observation flows over arbitrarily long time frame. Another problem with this approach is that it is not obvious how to check if an activity should be clustered or is fine as it is.

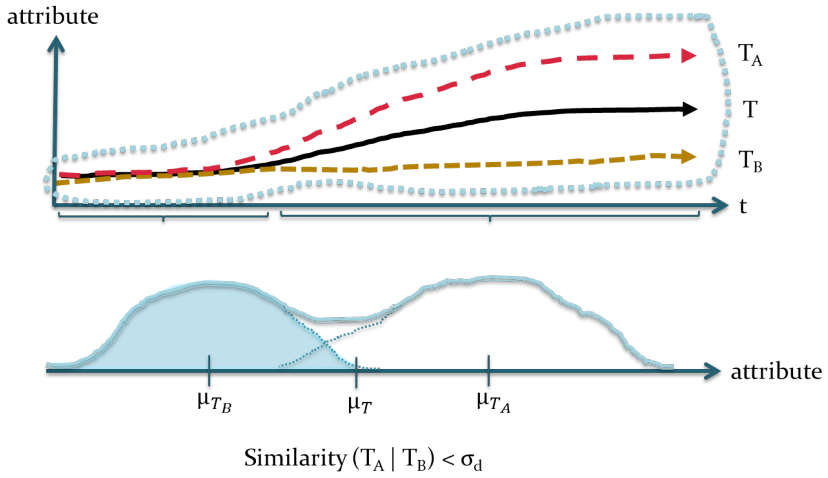


Figure 5.9: *The wide activity with trajectory T actually consists of two smaller activities with trajectory T_A and T_B. In the lower figure it is shown that the similarity between the two smaller sub-activities is below the threshold σ_d which indicate that the wider activity should be split into these two smaller activities.*

A possible alternative solution is to have a hierarchy of detail levels, illustrated in Figure 5.10, for each activity. Sub-activities are modeled in the same way as the ordinary primary activities are, but with a similarity threshold that is lower than on the detail level above them. If the primary detail level has a threshold determined by $s_{threshold}$, then the detail level below might have a threshold that is $\frac{s_{threshold}}{2}$. The lower similarity threshold allows two more activities to be formed side by side, without them being considered the same or close as to warrant a merge. If two activities on this level would still be separated with the lower detail of $s_{threshold}$ then the activity on the primary detail level should be split to reflect the separation of the two separated activities on the level below. The hierarchy does not need to be particular deep, with a depth adjustable by the user for increasing the cluster resolution for the cost of less computational performance. This solution was not investigated within this work and further elaboration end experiments are left to future work.

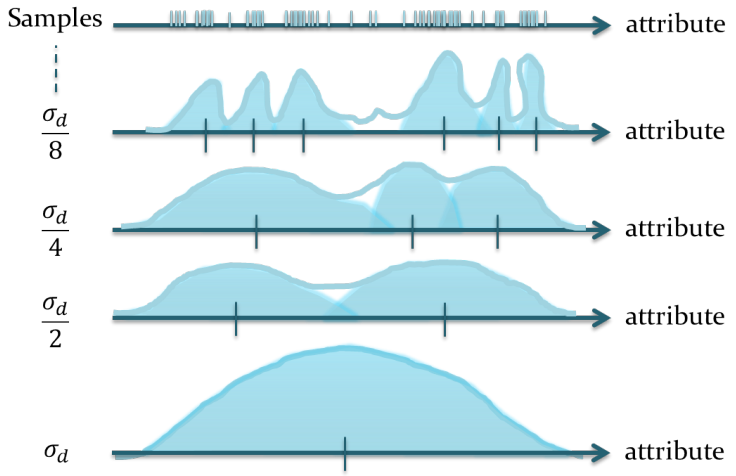


Figure 5.10: *A detail hierarchy of an activity. The primary activity is at the bottom with detail level σ_d. The clusters of activities given different detail levels are shown above it until the detail level is so low that only the actually observed samples are seen.*

5.3 Activity Recognition and Prediction

Activity recognition and prediction are considered both within the span of single activities, and over chains of activities in the Activity Transition Graph. In the former case is the continuous activity model of activities considered and in the latter case is activities viewed as discrete nodes in a network of activity transitions.

This division is motivated by the fact that we have two different kinds of models; the activity model which models the variable state for the duration of a single activity, and the Activity Transition Graph that models the likelihoods of transitions between activities. The latter assumes that we know which activity we are in, in order to predict the likelihood for future and past activities. In comparison, the activity model is used to classify sections of an observed trajectory in order to find the most likely activity of each section. For example, consider the car in the T-crossing scenario (Figure 1.1, bottom to the right). The car will be classified as performing the initial activity in the beginning of the entry of the T-crossing. The only information available at this stage about if the car will turn or not is from the transition statistics of the past, stored in the Activity Transition Graph. Not until the car gets close to the point of intersection, where the first activity branches into a turning activity and a straight activity, can we say more about the likelihood for a turn. At this stage the primary concern is which of the two activities best explains the observed behavior. The uncertainty will be high with only the first observed samples directly after the intersection point, but the classification grows more certain with the car making its way and covering more and more of the length of either of the activities. From the start on the other

side of the transition point until the object has covered an entire activity can we perform run-time early classification, with a final classification afterwards. This means that the framework can always supply the currently best activity classification, and the currently most likely future activity transitions can be predicted as a consequence. Figure 5.11 illustrates the different cases of predictions.

Figure 5.11: *Activities are nodes and the arrows are possible transitions between nodes. The black node is the current activity. The nodes with question marks are the nodes of possible future or past activities. (a-b) concern prediction of future activity. (c-d) concern prediction of past activity. (a) and (c) are predictions from within the current activity (the possible next/previous activities are evaluated using the local likelihood while the objects activity is still best explained by the current node) weighted with the transition probabilities. (b) and (d) are predicted based only on the transition probabilities of transition between the intermediate activity and the nodes marked with question marks.*

5.3.1 Activity Recognition and Prediction

For a collection of observed data points, we want to find out which activity that best explains these observations (Activity Recognition). We also want to predict expected future observation data points, given that we have found a most likely activity for our observations (Prediction).

The observed data points can either be treated separately, or be modeled using a Gaussian process to account for their dependence and that they are samples of an underlying trajectory. The local likelihood (Equation 4.5) can be calculated for either case, but if we do not model the observations as a GP we can only be as accurate as the available observed data points are when they individually have their local likelihood calculated. If they are modeled as a GP the local likelihood can be calculated for many more points, since the GP model allows the observed data to be extrapolated by estimating the underlying fluent.

We expect the observations to be iteratively available (as a stream) and there exist efficient ways to iteratively include new data points in Gaussian process models without having to recalculate the entire matrix inverse of all the data points (Smith et al., 2012).

The local likelihood of the observations given the various activities deemed relevant can be calculated for a single point, for a limited interval or for the entire up-till-now known interval of each activity. These different calculations provide

different information, such as which activity that best explains a given observation, or that best explains the observations over a specific interval. This is illustrated in Figure 5.12, where an observed trajectory is compared to a known activity.

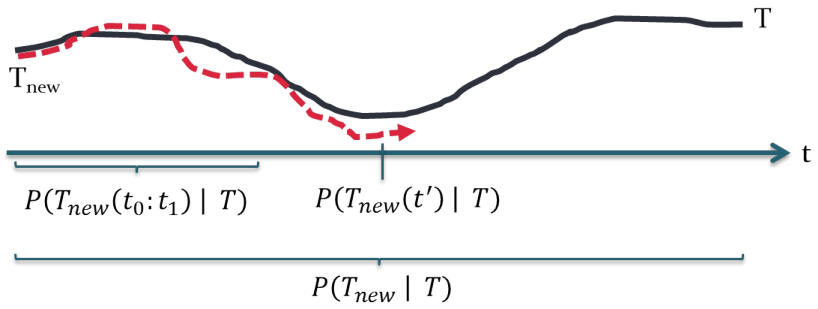


Figure 5.12: *The different possible probability calculations of a new observed trajectory T_{new} given the activity T, over the parametrized time t of the activity.*

The comparison of the local likelihood of the observations between different activities decide which activity that is most likely to explain the observed data. If the observed data is not sufficient yet to be compared to the entire activity, the remainder of the activity can be used to predict the expected continuity of the observed data. The Gaussian process is a generative model, so an entire instance of an activity trajectory or simply the continuation of the observed trajectory can be generated. An intuition to the classification an observed trajectory together with the prediction as a consequence of the classification is shown in Figure 5.13.

5.3.2 Activity Chain Recognition and Prediction

The State Space Graph and the Activity Transition Graph can be used in several ways to detect and predict activities. There is statistical information such as the evidence for any activity, similarity between any two activities and the probability of transition between activities. Assuming the presence of one or several 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.

The most likely path from activity A_i to activity A_j is the shortest path in terms of the negative log of the transition probabilities in the Activity Transition Graph. This follows from the fact that the likelihood of any given chain of activities equals the product of the transition likelihoods of that chain. The negative log likelihood of this path, $-\log P(A_j|A_i)$, is the path cost. Because of this, general graph algorithms for finding a shortest path can be used out of the box.

The most likely path from any activity to an activity at most n activities away,

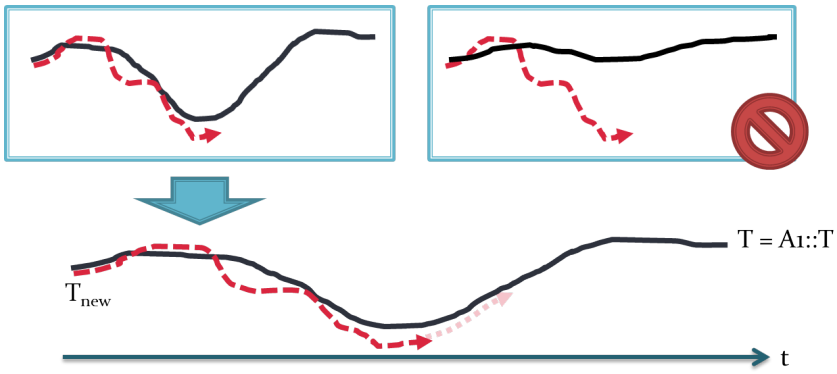


Figure 5.13: *The observed trajectory T_{new} has been calculated to be more likely explained by the activity in the left box than in the right box. The bottom graph shows the prediction of the observed trajectory given that the currently most likely activity is indeed able to best explain the continuation of the observed trajectory.*

r meters away, l meters travelled or any similar constraint is solved for in the same way as before with the added restriction on allowed nodes to expand in the search. Prediction forward in time and backwards in time is possible by following the causal arrows of the Activity Transition Graph forward or backwards respectively.

More dynamic information regarding the learning procedure is also available. Events can be created of when the activity learning module creates, updates and merges activities, or when the classification of an object is finished due to the object crossing an intersection point. Activities can be classified in real time, and their most likely chain backwards and forwards in time can be found to cope with unobserved behavior. The k most probable activities can be streamed for every observed object, or the most likely activity can be used for event generation by thresholding the probability of the activity having occurred. Finally the probability of an object being in a transition or performing an activity can also be deduced, including the probability of the performed activity being abnormal.

6

Experiments and Result

To verify that our approach works as expected, we have performed 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 from physical platforms.

6.1 T-Crossing Case Study

The T-crossing scenario is similar to Figure 1.1 on page 4 where two types of trajectories are observed: straight trajectories and left turning trajectories. In each of the experiments we used 50 of each type. Trajectories are provided to the framework in a random order. Three different state spaces are compared:

$$\text{Case } 1: \quad y = \{position_{\hat{x}}, position_{\hat{y}}\}, \qquad y \in \mathbb{R}^2 \tag{6.1}$$

$$\text{Case } 2: \quad y = \{position_{\hat{x}}, position_{\hat{y}}, heading_{\hat{x}}, heading_{\hat{y}}\}, \qquad y \in \mathbb{R}^4 \tag{6.2}$$

$$\text{Case } 3: \quad y = \{position_{\hat{x}}, position_{\hat{y}}, velocity_{\hat{x}}, velocity_{\hat{y}}\}, \qquad y \in \mathbb{R}^4 \tag{6.3}$$

The headings in the second case are the normalized velocities from case three. The converged trajectories of the first and second case can be seen in Figure 6.1 where the generated data is also shown.

With position and headings (normalized velocities) the turning activity begin earlier than with only positions available, and even more so with the slow down taking place. The framework learns a turn activity earlier when having positions and headings (case 2) compared to when only having positions (case 1). As a consequence it is possible to predict that a car will turn earlier. Similarly, using both velocity and position (case 3) the framework is capable of learning the turning

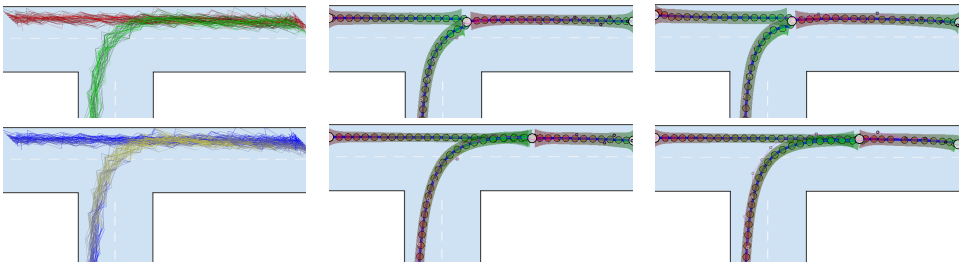


Figure 6.1: *Case study. Top row: Scenario with positions only. Bottom row: Scenario with a slow down. The figures in the middle show the three learned activities after two observed trajectories, while the figures to the right show the result after all observations. Blue color indicates a higher speed than yellow.*

activity even earlier than when using headings (case 2), and can therefore predict turn activities even earlier.

6.2 UAV Case Study

The second experiment is a quadcopter UAV flying various patterns through a four times four meter large grid with a top speed of $0.5 m/s$. The UAV always flies in the same direction between each pair of points (Figure 6.2, Left). The branches in a crossing have different probabilities. The experiment evaluates the framework using 72 simulated flight trajectories and 25 real flight trajectories.

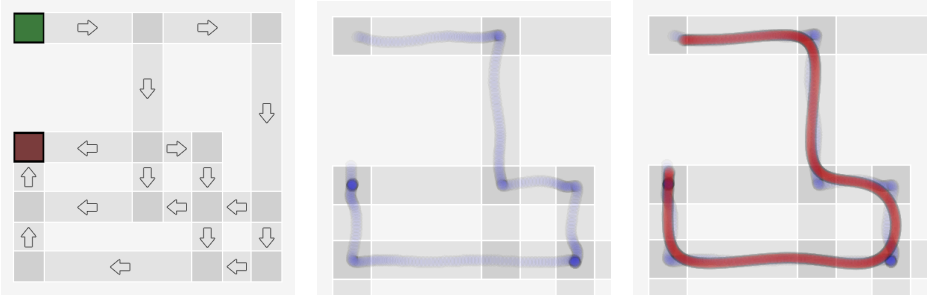


Figure 6.2: *(Left) The graph of flight patterns used for the second experiment. A UAV always enters from the top-left green node and then arbitrarily traverses the graph until it reaches the middle-left red node. (Middle) An example trajectory with the positions of the UAV shown as blue circles. At 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 consists of full trajectories of 2D positions. All preprocessing and processing required is included in the presented times. 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. The minimum distance between two data points was set to $15cm$. This imposes a limitation to the resolution of detectable details, which should be a minor issue. It does however reduce the influence 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 6.2.

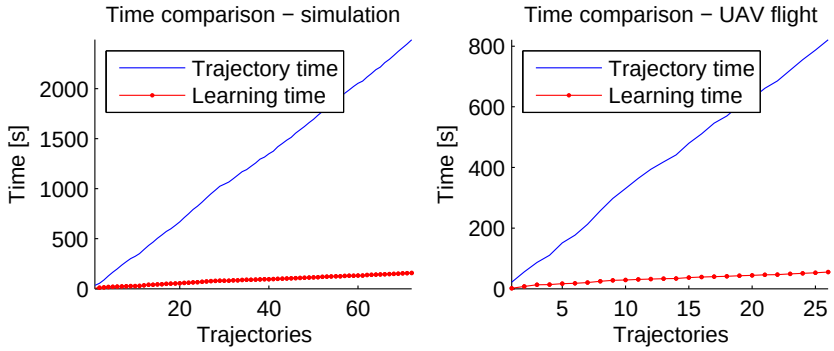


Figure 6.3: *A comparison between the accumulative flight time and the accumulated learning time after a given number of observed trajectories.*

A comparison of the flight time of trajectories and the learning time of the trajectories is shown in Figure 6.3. The experiment ran on a single 2.5 Ghz Gen.4 Intel i5 core. The learning rate of the simulated trajectories is 11.89x real-time and the corresponding learning rate of the real flight trajectories is 10.39x real-time. 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.

Figure 6.4 shows that the most computationally heavy work in the framework is that of estimating a GP model of a atomic trajectory and re-sampling GP models using **SGPUS**, where the former is performed in *pre-process trajectory* and the latter in the create/update/merge operations. These two calculations have complexities of $\mathcal{O}(M^3)$ and $\mathcal{O}(M^2 + N^3)$ in the number of observed data points M the trajectory consists of and the number of support points N used by the activity models. It takes longer time to create GP models of long observed trajectories compared to short observed trajectories. At the start of a system with no prior knowledge, no activities are known and the first learned activity will be the first observed trajectory. Any successively learned activity will either be added as the whole observed trajectory, or contribute to the splitting of activities and thereby

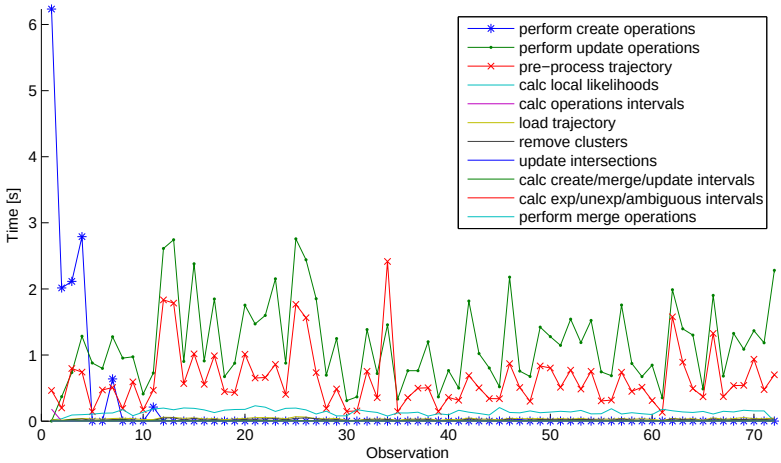


Figure 6.4: *The legend is sorted on maximum processing time and the three most costly tasks have distinct markings. The fourth most costly task is to calculate the local likelihoods for a new observation and it is the only clearly visible function apart from the three most costly.*

make known activities shorter. As a consequence, the time it takes to learn new activities will generally start high and later converge to a lower time as the underlying structure has been more or less learned and most activities are as short as they ever will be. This can be seen in Figure 6.5, where the time is normalized on the number of samples of each trajectory to remove the effect that long observed trajectories take longer time to model than short observed trajectories.

The data set has exact positions for the UAV both from a Vicon vision system and in the simulation. We introduce the assumption of observation noise in order to cluster the different trajectories. The assumed the observation noise to be higher for the real flight data to allow for larger variations within the initial activities due to the lack of merging operations. We observed that the constraints on and calculations of which intervals to merge and not to merge, close to intersections, is not always sufficient, merge operations where therefore omitted in this experiment

The State Space Graph at various stages of the experiment is shown in Figure 6.6 and Figure 6.7. The resulting models display a corresponding description of the observed trajectories with a high model quality. The resulting state space model of the real flight trajectories capture several parallel activities that were scheduled for merge but were not since we disabled that operation. This resulting model also highlighted a need for a tighter lock of the endings of activities and their intersection points, such as in the case of the top-middle node.

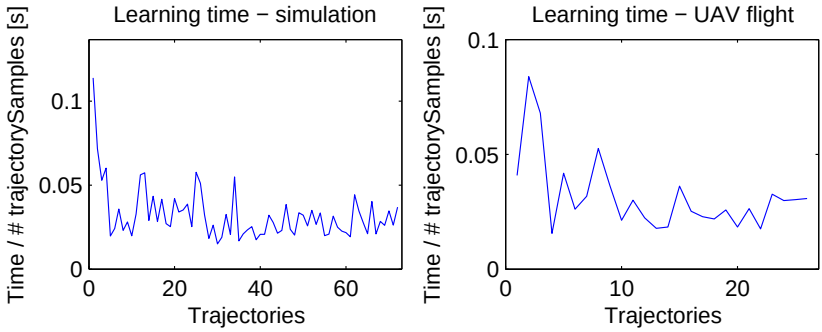
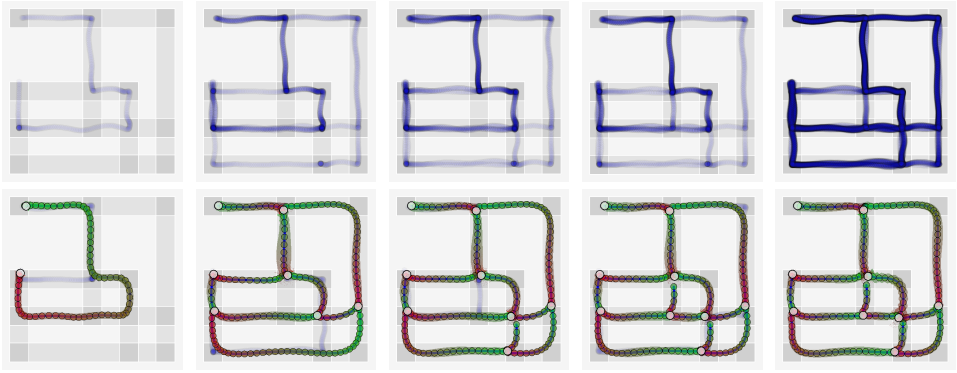
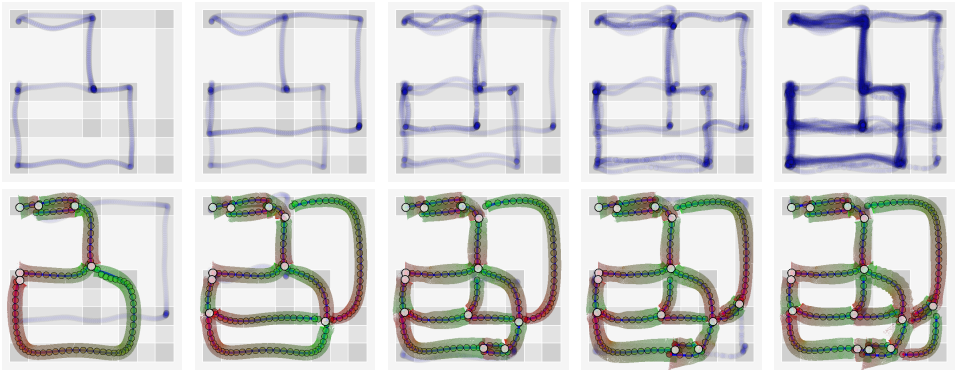


Figure 6.5: *The learning time of the framework for simulated respective real flight after a number of trajectories. The time is normalized using the number of samples of the observed trajectory to illustrate that the time cost shrinks with the the learning of new activities and stabilizes when all activities have been learned. The normalization is performed since Gaussian process regression is cubic to the number of data points, so the learning takes longer with longer observed trajectories. The x-axis indicates how many activities that has been processed so far.*



(a) *1 path* ***(b)*** *6 paths* ***(c)*** *10 paths* ***(d)*** *11 paths* ***(e)*** *72 paths*

Figure 6.6: *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. The bottom row also shows the next observed trajectory in blue. The color of each activity transitions from green at its start point to red at its end point.*



(a) 2 paths *(b) 3 paths* *(c) 6 paths* *(d) 8 paths* *(e) 25 paths*

Figure 6.7: *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. The bottom row also shows the next observed trajectory in blue. The color of each activity transitions from green at its start point to red at its end point.*

7

Discussion and Conclusion

Learning and recognizing spatio-temporal activities from streams of observations is a central problem in artificial intelligence. In this thesis, we have presented an unsupervised stream processing framework that learns a probabilistic and continuous representation of observed spatio-temporal activities and their causal relations from observed trajectories of objects. It provides a fundamental layer to build from for continuous maintenance and integration of high-level and low-level knowledge by combining symbolic and quantitative models to archive learning, recognition, prediction and simulation on the presented abstraction levels. The framework has been evaluated in two different experiments involving both simulated and real data.

This chapter starts with a general discussion and overview of the framework. The section after covers an algorithmic complexity analysis of the framework and the **SGPUS** algorithm in comparisons with alternative techniques. After this an extensive analysis of the qualitative performance of the abilities of the framework and the experimental results is presented in the subsequent section. Finally, this thesis is concluded with a conclusion and a final section about future possibilities building upon this work and where to go from here on.

7.1 The Framework

The aim of the framework is to provide a system with the ability to detect and predict dynamic normative behaviors from observations. It is especially designed with the aim of running in real time, and to be able to perform unsupervised learning of new behaviors and updating the models. Everything is modeled using probabilistic approaches, so a confidence will be available in every regard.

The system's understanding of the world is in terms of activities and causal relations between the activities, where activities are trajectories in the state space. The state space consists of a Euclidean space containing all state properties of interest, such as for example positions, velocities, redness, humanness and so on. Any property that is continuous can be used. Discrete properties can be used, although it is not recommended, and they will be made fuzzy in their representation since the model is continuous and values may be interpolated or extrapolated. A conversion back to a discrete value is left to the user of the framework, e.g. by thresholding. The models used can be constructed by humans and updated fully supervised. However, the strength of letting the framework run and learn unsupervised **[D12]** is that the system can then adjust its understanding of the world when what is observed diverges from what its model expects.

The framework currently tries to learn the minimal unique set of atomic activities, and the causal relation of these. The atomic activities are modeled using Gaussian Processes, which provide a continuous trajectory through the state space, with a certainty at every point in the state space. The certainty, or variance of the Gaussian distribution, has a dual interpretation and purpose. When an activity model is supported by very few observations, it is still largely uncertain if it captures the observed activity or not. An activity model captures the observed activity if instances of that observed activity are always explained by the model with high enough probability/confidence. When an activity model has accumulated a very large amount of observations, the confidence of the model is expected to be higher, and the variance can more and more be seen as the width of the modeled trajectory. This means that we expect observations of the activity to fall within the variance of the model such that the probability/confidence of the model explaining the observations is high enough. Our current way to achieved this is to weight the likelihood of the different activities according to their evidence, where the evidence is the number of observed trajectories an activity has explained so far. The activity model is limited by the fact that it cannot model a trajectory which intersect itself. This is currently assumed to not happen, but an easy work-around is to split any such observations into non-intersecting pieces.

The context learned by the presented framework is the relations between the activities, and this context is modeled using two graphs. The Activity Transition Graph is a Markov-chain, where each node is an atomic activity and the edges are temporally directed transitions between the activities. Each edge has a transition probability, which is learned by observing objects transitioning in the graph. This graph allows for reasoning over the causal relations of activities **[D9]**. The second graph is called the State Space Graph, and it is a directed graph connecting the activity models spatially, where the edges are activities and the nodes are intersection points **[D8]**. These two graphs are both learned at the same time by creating, merging, separating and updating activities based on observations. An activity not explained well enough by the model, according to a user parameter, is deemed as novel/unknown **[D10]** and created as a new activity **[D1]**. Activities that are too similar, utilizing the same measurement, are merged. The separation functionality is not yet a part of the current framework, which introduces

some limitations that make the framework less invariant to the order in which the observations are learned. Observed trajectories are recognized as chains of activities **[D6]**, and recognized activities are updated with the matching interval of the observed trajectory **[D2]**. The size of the activity model does not increase as a consequence of the number of observed trajectories and only depend on the structural information provided by these observations. This allows for an endless number of integrated observations without performance bottlenecks **[D5]**.

Trajectories through the State Space Graph can be made continuous by applying fusion on the activities involved; this provides a resulting continuous trajectory similar to that of a single Gaussian Process. The segmentation of atomic activities into short sparse Gaussian Processes makes the use of Gaussian processes feasible in large real world scenarios. It also makes real-time performance feasible in practice for the learning and updating of the Gaussian Processes models used for modeling activities. Since Gaussian processes are generative probabilistic models we can generate example data in the form of trajectories with arbitrarily high resolution for any activity or activity-chain. This means that we can generate data sets with the same statistical properties, i.e. regarding activity transitions and activity width, and on the same form as the observations used to learn models in the frameworks knowledge base **[D8]**. This allows for prediction both within individual activities and over chains of activities **[D7]** both forwards and backwards in time **[D11]**.

7.2 Computational Complexity and Performance Factors

The framework is dependent on various parameters which effect its computational and qualitative performance. The number of support points N of each activity is determined by a minimal distance $minDistance$ between each support point, so the actual number of support points of an activity is dependent on its length L:

$$N = \frac{L}{minDistance} \tag{7.1}$$

The hyper parameter optimization of a Gaussian process has a complexity of $\mathcal{O}(M^3)$ (Rasmussen, 2006) in the number of support points, which is currently performed on every new observed trajectory. This could potentially be replaced by sparse GPs (Snelson and Ghahramani, 2005, 2007) which have a complexity of $\mathcal{O}(K^2M)$ where $K < M$, M is the original number of support points and K is the number of sparser support points. The GP model of the observed trajectory is used by the **SGPUS** algorithm to make the GP sparse or for combining multiple GPs in updating and merging operations of the framework. **SGPUS** has a complexity of $\mathcal{O}(NM^2 + N^3)$ where M is the original number of support points and N is the new number of support points, which is usually much lower than M. N is determined by Equation 7.1 but can be bounded by having a strict maximum number of support points N or a strict maximum length L. Although not

investigated here, it should be possible to have activities longer than a constant length split into a chain of sub-activities and treated as a single GP by **SGPUS** by utilizing fusion as described in Section 3.3. As a consequence the computational cost could potentially be greatly reduced by a more systematic utilization of local GP models. This would however introduce additional approximations since Gaussian process models are only influenced by their own support points. For example, a split in the middle of a turning trajectory might not provide two well-connected half-turns since neither model will be aware of the others continuation of the turn. Consequently, care has to be taken when selecting the points to split activities to sub-activities.

The strength of the **SGPUS** compared to other sparse GP techniques is that it has a very low complexity when incremental in its inclusion of new data. **SGPUS** has a higher complexity in the number of data points M with a complexity of $\mathcal{O}(NM^2 + N^3)$ compared to $\mathcal{O}(MN^2)$. However if the data points come in clusters that have a uniform distribution over the input variable (e.g. similar trajectories) then these can be included cluster after cluster making the variable M depend on each individual cluster size rather than total number of data points. A consequence of this is that **SGPUS** has a complexity that is *linear in the number of observed trajectories*. To make such an iterative approach work we have to assume that all (or at least most) trajectories used by **SGPUS** contain the structure that we want to capture. This is essential in order for the final sparse GP model to converge towards this structure. To make a summary of the limitations: the number of support points must be high enough to capture the sought structure, and each input (each trajectory) must contain enough of the sought structure. In the second experiment with the UAV scenario we downsampled the observed trajectories. If the down sampling would have been too large, then we might have been unable to model all the interesting structures since these structures would not be contained sufficiently in the used input data to **SGPUS**.

The iterative approach, of learning one trajectory after another, does introduce additional problems and approximations. Specifically the convergence seems to be slower than if all data is used at once. That the convergence can be slower is however not a matter of too much concern, since data in most cases will be in abundance. The models are built to be able to assimilate large numbers of observations, as expected with systems working continuously for long periods of time in the real world. The effect of the convergence on the quality of the State Space Graph, and that of reproducibility and completeness, is however a matter that has to be investigated further, together with the stability and robustness of convergence of **SGPUS**.

There are parameters that let the user adjust various parts of the framework between on one hand slower but more exact calculations, and more approximative but faster on the other. **SGPUS** uses a parameter that specifies the maximum number of iterations, where each iteration improves the result more or less. The calculation of the length of a GP is performed using an approximate Riemann integral, where a parameter sets the step length. The shorter step length, the more

exact length we get. The length of a GP is used similar to the way L is used in Equation 7.1 or the length on an activity used to compare with $l_{threshold}$ in Section 5.2. The local likelihood in Equation 4.5 is calculated at points on an activity with a certain distance between each point, as used in Section 5.2. The shorter distance, the higher resolution but also the slower is the computations.

Each operation of the framework is linear in the number of activities, as the local likelihood (Equation 4.5) is currently calculated for all known activities. An octree or similar data structure should make it possible to reduce the number of activities necessary to compare with, if the underlying structure is somewhat sparse. The framework is also linear in the dimension of the state space if the dimensions are assumed independent as they are in this work. This is the case for both mappings to and from time parameter and attributes of the activity model. In the case of the time-to-attribute mapping there is in effect one GP for each attribute, which corresponds to one dimension. In the case of the attribute-to-time mapping the squared exponential kernel used (Equation 2.8) is internally the function of the $\mathcal{L}^2$-norm of two attribute vectors, which is a linear operation that produces a scalar. If the dimensions of the state space are not assumed to be independent the complexity is quadratic.

7.3 Discussion

A higher dimensional state space makes it easier to discriminate between observations of different activities. As a consequence, more activities can be learned, detected and predicted. The increased ability to discriminate activities is illustrated in the T-crossing experiment (section 6.1), where the use of positions and velocities allow the framework to differentiate between turning and not turning much earlier than with only positions. With an assumption of independence between the dimensions, the computational complexity of the framework and its operations is linear in the number of dimensions of the state space with a low constant. As a consequence, it is encouraged to include all available properties (directly or indirectly observable) since this significantly improves the framework's activity discrimination potential. By treating dependent variables as independent, each variable has to be modeled on its own which can lead to larger uncertainty. Treating them in accordance to their actual dependencies has some possible advantages, such as more true probability density shapes of the activity models and a stronger ability to compensate for a lack of data or too few support points. The time complexity is however squared in the number of dimensions of the state space without the independence assumptions, so this will need more consideration and evaluations in future work. It is important to scale the dimensions in different ways when using kernel weight from equation 2.10 since then only a single kernel parameter will determine the smoothness of all of the dimensions. This scaling can be hard to know beforehand and it is therefore recommended to use weights from equation 2.11, which let the software optimize the smoothness parameters of each dimension.

We show in the UAV flight pattern experiment (section 6.2) that our framework is capable of learning observed activities in real-time **[D4]** using real-world data **[D3]**. Figure 6.5 indicate that the framework's performance can increase by learning new activities. The performance of the framework is limited by two factors: the GP modeling of each observed trajectory and the **SGPUS** calculations during create/update/merge operations. This is supported by both the computational complexities of the operations as described in section 7.2 and by the empirical results of the experiments (Figure 6.4). In the figure we can see that the incorporation of new observations into known activities (perform update operations) and GP modeling of the observed trajectory (pre-processing trajectory) are the two tasks that take the most time. The create operation takes a long time because it might have to split activities, which requires running **SGPUS** for each resulting activity. The current implementation of the create operation does not make all splits simultaneously, but instead does it recursive and therefore performs a lot of extra **SGPUS** calculations. Both pre-process and create/update/merge have a computational time that depends on (1) The number of points as input (computational complexity) and (2) The number of iteration of the GP optimization and of the **SGPUS** (a large computational complexity constant). The former can be reduced by making the pre-process clustering of individual observed samples larger with the effect that less observation structure will be available. The latter can be lowered by adjusting the parameter of maximum number of iterations or changing the stopping criteria, with the effect that the model of the observed data will be mer approximate.

It is in general the case that the performance of the framework has potential to decrease over time due to two different processes:

1. An increase in the number of shortened activities due to additional intersections, by the learning of new activities.

2. The increased resistance to change of increasingly well supported activities, by the number of incorporated observed trajectories.

The first process increases the performance since shorter activities means shorter Gaussian process models. If the structure of the activity transitions is dense then there will be many short activities rather than a few long. So if there is a domain structure that contains many activity transitions, then the performance will gradually increase with the learning of new activities as these will split longer activities into shorter activities. This affects the create, merge and update operations. The second process can increase performance since the parameters optimized in **SGPUS** will start closer and closer to their targeted optima the smaller change that is required by a well supported activity in update operations. This is the case since the combining described in section 3.3 weights the combined GPs in accordance to their amount of evidence, and a new observed trajectory has only 1 as evidence amount while an already learned activity can have arbitrarily high evidence. The variable step-length of the gradient descent used by **SGPUS** can be made to take advantage of this, so that the number of iterations required in the gradient descent can be decreased significantly. What this means is that more

observed trajectories makes the update operations cheaper.

A generally better adaptive step-length for gradient descent in **SGPUS** would also make it faster. It has been observed that the gradient steps of the observation noise vector $\sigma^2_{1...N}$ and kernel parameter θ_1 get stuck in a competing position close of convergence and therefore does not let the stop criterion of the gradient descent be fulfilled, with the consequence that it will run for the maximum number of iterations. The current step-length adaptation of θ_1 might not be consistently stable either if two local optima are close by since the doubling and halfing of the step-length might make it step over into the other. There also exist much faster non-linear optimization methods which can make use of the Hessian or higher gradients.

Analytical integration of a Gaussian process over its range is another speed improvement that should be possible to make. It would make the calculation of activity lengths much faster and more accurate, which is used by the create, update and merge operations, and it would replace the Riemann sum currently used. It should also be possible to perform closed form analytical integration over products of Gaussian processes over their range, making it possible to calculate the KL-divergence (Rasmussen, 2006) or Bhattacharyya distance (Schweppe, 1967) between activities. This could be an even better similarity measures between activities than what is currently used, since they are continuous rather than sampled and reflect theoretical properties more in line with what is actually sought after in the notion of similarity between probabilistic distributions. These similarity measures are defined for Gaussian processes with the same number of support points by using the duality of Gaussian processes and Gaussian distributions. To extensively use **SGPUS** to temporarily re-sample activities for similarity comparison is however quite time consuming compared to closed form analytic evaluations of the similarity measures mentioned. These calculations should be bounded by the complexity of the compared activities' mean and variance functions (equation 3.12-3.13) with a complexity of $\mathcal{O}(N^2)$ in the number of support points of each activity, and with a very small constant for the evaluation of the analytical expression. An additional advantage of using a continuous similarity measure of the activities' GPs is the increased accuracy (no longer an approximation) and that it could provide a highly efficient way to compare activities in the future extensions of complex activities and activity equivalence classes.

It is common for trajectory clustering methods to either use average trajectories (e.g. centroids), cluster prototypes (e.g. a random trajectory from the set) or envelopes (i.e. linear splines between a set of points or a set of Gaussian distributions) (Morris and Trivedi, 2013). Our activity model provides a continuous probabilistic envelope capable of providing a Gaussian distribution for each real point along the activity mean-line, from start to end of the activity. It incorporates the probabilistic nature of groups of individual trajectories that can be both noisy and be of various quality (such as be of different sample rate or contain missing data).

Another difference to common methods in trajectory clustering is our use of tra-

jectory normalization without loss of behavioral semantics accuracy, and the use of the inverse mapping for the activity similarity calculations. The loss of time-dependent information, such as velocity and acceleration, is a common effect of simple normalization techniques (Morris and Trivedi, 2013) and we solve this by explicitly having velocities and other properties as dimensions in the state space. This makes the interesting time-dependent properties invariant in regards to the normalization of activities' time parametrization ($t \in [0.0\ 1.0]$) and of re-sampling. Extending the dimensionality of the state space does not limit the performance in a significant way since the framework's operations are linear in the number of dimensions of the state space. These properties can for example be estimated from the observed data points by applying a Kalman filter. The normalization in time parametrization solves the issue of comparing trajectories of unequal length (Morris and Trivedi, 2013). Additionally, our use of the inverse mapping in the activity model let us compare overlapping parts of trajectories to each other.

The inverse mapping used by us have some limitations. The mapping is modeled using a Gaussian random field with a Gaussian kernel and is therefore localized as the super position of a set of multivariate Gaussian distributions in the state space. This means that a radial projection of a state space point onto the mean function of an activity is only accurately performed when the state space point is close enough to the activity. If it is not, then the projection is biased towards the mean state of the mean function of the activity. We would like to find a better solution that allows an inverse mapping that always project a state space point to the closest radial point on the mean function. A general analytic formulation of this does however seem to be impossible to find in the general sense since it is an equation of a sum of different exponential functions.

The key to our on-line learning approach is that new structures in the form of novel activities are incorporated in the knowledge base directly upon observation **[D10]**, compared to off-line approaches that start with all data and searches for structure. We perform a bottom-up clustering. Off-line cluster methods usually need to perform a pairwise comparison of all trajectories which is very computational expensive due to the quadratic complexity in the number of trajectories which can be very large (Morris and Trivedi, 2013). Our on-line method in comparison has a linear complexity in the number of activity models for adding a new observed trajectory.

The on-line approach provide several benefits such as adaptivity and reduced processing time but it also has some problems which need to be considered. The first main problem is that of reproducibility, meaning if the model will converge towards the same result regardless of the ordering of the trajectories observed. The other problem is that of the quality of the solution, meaning if the end result will find a global solution or a local solution and how the quality may vary. The most striking current shortcoming on reproducibility is described in section 5.2.4, with an example in Figure 5.9. We present a possible solution using a detail hierarchy of local sub-activity models for each activity model. Each level in the

hierarchy is supposed to capture a specific level of detail so that the best possible candidate clusters for an activity to split into is available without running any cluster algorithm. if this is a feasible solution and how much it would impact performance is however unknown and left for future work.

There exist a lot of work regarding traffic behavior understanding (Morris and Trivedi, 2013) and we would like to compare our performance on common data sets in future work. Our framework do have some issues in regard to this domain that has to be solved in the future. We do not yet explicitly support activities that are static in the state space, e.g. *to stand still*. Neither is the support for example multiple lanes in the traffic domain very good, at least not for modeling that cars can change lanes at arbitrary points. The contextual structure to support this is not yet a part of our framework and we view this as future work.

7.4 Conclusion

Learning and recognizing spatio-temporal activities from streams of observations is a central problem in artificial intelligence. In this thesis, we have proposed an unsupervised stream processing framework that learns a probabilistic and continuous 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.

The experiments show that our framework is capable of learning observed activities on-line in real-time using real-world data. The framework performs on-shot learning, meaning that a single observed trajectory is enough to learn a new activity model, which can be used for immediate recognition, prediction and simulation of activities and activity-chains. Due to the low complexity of the algorithms the solution is scalable to real-world situations. The approach has a time complexity for activity learning and recognition that is linear in the number of learned activities, in the number of observed trajectories and in the dimension of the state space. The learning of each activity's GP model is cubic in the number of supporting points. However, by keeping a strict maximum number of supporting points, the complexity can be bounded.

The framework provides a fundamental layer to build from for continuous maintenance and integration of high-level and low-level information. It combines symbolic and quantitative models to archive learning, recognition, prediction and simulation on multiple abstraction levels. It is capable of doing this on the current levels of abstractions consisting of the low-level atomic activity models and the higher level of the State-space Graph and Activity Transition Graph, where

activities are viewed as discrete states and the activity transitions are in focus.

The models used for knowledge representation provide an easy way for humans to understand the knowledge structure, and to add, remove or edit the knowledge base at any time. This is especially useful in domains where experts can provide prior knowledge. Since the framework have the ability of unsupervised adaptive domain knowledge acquisition and maintenance, these priors do not have to be entirely correct and will be modified and possibly extended by the framework, if allowed, as new information is observed.

7.5 Future Work

We would like to find statistically significant equivalent classes of activities, meaning activities that are similar to each other in different regards. There are many ways activities can be similar, but two main categories of similarity can be discerned: Similarity based on activity models (i.e. how the trajectory looks like and which state variables that are active or statistically significant) and similarity based on contexts such as activity transition configurations (i.e. activities of one kind follow after activities of a second kind leading to activities of a third kind.). This distinction between the two categories is in a sense connected to how we might expect to derive the classes of respective category. Classes in the former category are most likely best found by various ways of correlating activity trajectories in a bottom-up way. The latter is most likely easiest to find by building on the previous category classes and correlating sequences of transitions of various classes of activities. Figure 7.1 illustrates what we are after with this next step. Activities here have a more crisp representation than what we are currently capable of.

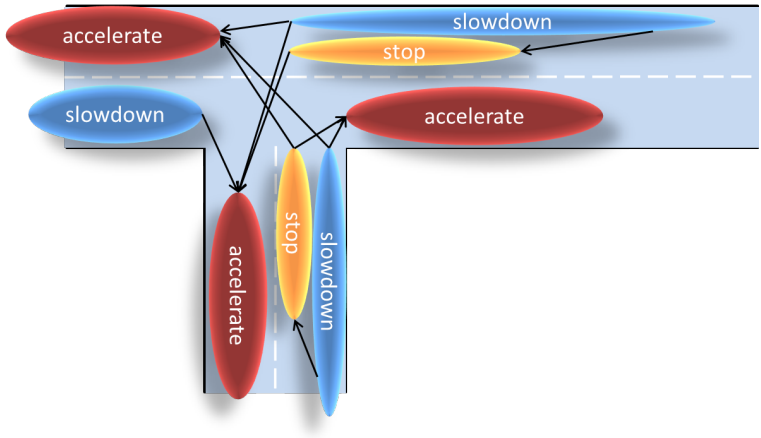


Figure 7.1: *Some of the expected activities and transitions between them that are expected to be found in a T-crossing.*

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. Currently complex activities are learned as chains of activities, but they are not treated as activities of their own. Regarding context there are at least two interesting aspects. First, how to generalize from activities in a particular situation, for example the activities in a particular T-crossing, to an activity in a wider range of situations, such as the activities in any T-crossing. This would correspond to learning the activities in a T-crossing context. A T-crossing context is different from a complex activity in that a context can contain disjoint activity graphs, for example in the crossing where no U-turns are expected and all the possible paths in and out of the crossing would be contained by the context. Second, how to take other objects and the state of the environment into account. The same observation could correspond to several different activities depending on this. For example, observing a car slowing down in a crossing could be either because it is going to turn, that there are other cars in front of it that are slowing down or because there is a red light. The framework is designed with these extensions in mind and provides the necessary building blocks.

A future prospect of the framework is to provide an efficient way for developing stream processing and robotic systems. A priori information can be introduced by humans constructing and editing models. The system can then refine the models using observations and learn new non-modeled behavior. These models can then again be edited by humans and let to be refined by the system once more. By letting humans labeling activities and contexts, the models and the framework can be integrated into stream processing or robotic solutions. A long term goal is to provide a reduction of feasible classes of activities and contexts to canonical forms, so that these can be stored and shared between systems and projects. A catalog of behaviors that can be used for any system to detect and predict, or even modify to better suit the current needs and environment of an application. These would then be modifications that still make it possible to communicate about the behaviors using a common language between agents and between agent and human.

We see this work as a step towards learning activities unsupervised for robotic systems to adapt to new circumstances autonomously and to learn new activities on the fly that can be detected and predicted immediately.

Bibliography

J. Aggarwal and M. Ryoo. Human activity analysis: A review. *ACM Comput. Surv.*, 43(3), 2011. Cited on page 3.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. ISBN 978-0-387-31073-2. Cited on page 52.

P. Boyle and M. Frean. Dependent gaussian processes. In *In Advances in Neural Information Processing Systems 17*, pages 217–224. MIT Press, 2005. Cited on page 15.

P. Brézillon. Context in problem solving: A survey. *The Knowledge Engineering Review*, 14:1–34, 1999. Cited on page 3.

S. Coradeschi. *Anchoring symbols to sensory data*. PhD thesis, Linköping UniversityLinköping University, Department of Computer and Information Science, The Institute of Technology, 1999. Cited on page 2.

G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, June 2012. ISSN 0360-0300. doi: 10.1145/2187671.2187677. Cited on pages 10 and 11.

M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg. Accurate scale estimation for robust visual tracking. *Proceedings of the British Machine Vision Conference BMVC*, 2014a. Cited on page 19.

M. Danelljan, F. Shahbaz Khan, M. Felsberg, and J. van de Weijer. Adaptive color attributes for real-time visual tracking. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2014 :*. IEEE, 2014b. Publication status: Accepted. Cited on page 19.

D. de Leng. Extending semantic matching in dyknow to handle indirectly-available streams, 2013. Cited on page 11.

C. Dousson and P. Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 324–329, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. Cited on page 11.

C. Dousson and P. L. Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *Proc. IJCAI*, 2007. Cited on page 3.

M. Endsley. Situation awareness global assessment technique (sagat). In *Aerospace and Electronics Conference, 1988. NAECON 1988., Proceedings of the IEEE 1988 National*, pages 789–795 vol.3, May 1988. doi: 10.1109/NAECON.1988.195097. Cited on pages 1 and 6.

R. Gerber and H.-H. Nagel. Representation of occurrences for road vehicle traffic. *Artificial Intelligence*, 172(4–5):351–391, 2008. Cited on page 3.

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. Cited on pages 15 and 40.

N. Guillarme and X. Lerouvreur. Unsupervised extraction of knowledge from S-AIS data for maritime situational awareness. In *Proc. FUSION*, 2013. Cited on page 18.

F. Gustafsson. *Statistical Sensor Fusion*. Studentlitteratur, 2010. ISBN 9789144054896. Cited on page 22.

F. Heintz. *DyKnow : A Stream-Based Knowledge Processing Middleware Framework*. PhD thesis, Linköping UniversityLinköping University, The Institute of Technology, KPLAB - Knowledge Processing Lab, 2009. Cited on page 11.

F. Heintz and P. Doherty. Dyknow : An approach to middleware for knowledge processing. *Journal of Intelligent & Fuzzy Systems*, 15(1):3–13, 2004. Cited on page 11.

F. Heintz, P. Rudol, and P. Doherty. From images to traffic behavior - a uav tracking and monitoring application. In *Proc. FUSION*, 2007. Cited on pages 3, 4, and 10.

F. Heintz, J. Kvarnström, and P. Doherty. Bridging the sense-reasoning gap: Dyknow - stream-based middleware for knowledge processing. *Advanced Engineering Informatics*, 24(1):14–26, 2010. Cited on page 1.

C. T. M. Keat and C. Laugier. Modelling smooth paths using gaussian processes. In *Proc. FSR*, 2007. Cited on page 13.

K. Kim, D. Lee, and I. Essa. Gaussian process regression flow for analysis of motion trajectories. In *Proc. ICCV*, 2011. Cited on pages 13, 18, 39, 40, 41, and 42.

S. Kim and J. Kim. Occupancy mapping and surface reconstruction using local gaussian processes with kinect sensors. *IEEE T. Cybernetics*, 43(5):1335–1346, 2013a. Cited on page 22.

S. Kim and J. Kim. Continuous occupancy maps using overlapping local gaussian processes. In *Proc. IROS*, 2013b. Cited on page 13.

D. Makris and T. Ellis. Learning semantic scene models from observing activity in visual surveillance. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(3):397–408, 2005. Cited on page 18.

B. T. Morris and M. M. Trivedi. Understanding vehicular traffic behavior from video: a survey of unsupervised approaches. *Journal of Electronic Imaging*, 22 (4):041113–041113, 2013. Cited on pages 4, 17, 18, 19, 71, 72, and 73.

H. Nyquist. Certain topics in telegraph transmission theory. *American Institute of Electrical Engineers, Transactions of the*, 47(2):617–644, April 1928. ISSN 0096-3860. doi: 10.1109/T-AIEE.1928.5055024. Cited on pages 28 and 34.

M. Osborne. *Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature*. PhD thesis, PhD thesis, University of Oxford, 2010. Cited on page 23.

C. Piciarelli and G. L. Foresti. On-line trajectory clustering for anomalous events detection. *Pattern Recognit. Lett*, pages 1835–1842, 2006. Cited on page 18.

C. E. Rasmussen. Gaussian processes for machine learning. MIT Press, 2006. Cited on pages 13, 17, 21, 67, and 71.

M. Schneider and W. Ertel. Robot learning by demonstration with local gaussian process regression. In *Proc. IROS*, 2010. Cited on pages 15, 21, and 22.

F. C. Schweppe. On the bhattacharyya distance and the divergence between gaussian processes. *Information and Control*, 11(4):373–395, 1967. Cited on page 71.

M. Smith, S. Reece, S. Roberts, and I. Rezek. Online maritime abnormality detection using gaussian processes and extreme value theory. In *Proc. ICDM*, 2012. Cited on pages 15 and 55.

M. Smith, S. Reece, I. Rezek, I. Psorakis, and S. Roberts. Maritime abnormality detection using gaussian processes. *Knowledge and Information Systems*, pages 1–26, 2013. Cited on page 13.

E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Proc. NIPS*, 2005. Cited on pages 15 and 67.

E. Snelson and Z. Ghahramani. Local and global sparse gaussian process approximations. In *Proc. AISTATS*, 2007. Cited on pages 15, 23, and 67.

M. Tiger and F. Heintz. Towards learning and classifying spatio-temporal activities in a stream processing framework. In *STAIRS 2014 : Proceedings of the 7th European Starting AI Researcher Symposium*, number 264 in Frontiers in Artificial Intelligence and Applications, pages 280–289, 2014. Cited on page 6.

J. Wang, D. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(2):283–298, 2008. Cited on page 13.

X. Wang, X. Ma, and E. Grimson. Unsupervised activity perception by hierarchical bayesian models. In *Proc. CVPR*, 2007. Cited on pages 17 and 19.

Appendix

A

Variance Estimation Derivation

In section 3.4.1 we formulate a least squares error function over the estimated variance function of the estimated Gaussian process (equation 3.17):

$$\mathcal{E}(\theta_1, \sigma_1^2, \ldots, \sigma_N^2) = \mathcal{E}(\phi) = \frac{1}{2M} \sum_{k=1}^{M} (\sigma_{est}^2(x_k|\phi) - \sigma_{obs}^2(x_k))^2 \tag{A.1}$$

We want to use this error function in the gradient descent, and to do so we must differentiate it with respect to the variables we want to optimize over. These are $\{\theta_1, \sigma_1^2, \ldots, \sigma_N^2\}$. The variance function in the expression is defined in equation 3.13 as

$$\sigma^2(x^*) = k(x^*, x^*) - k(x^*, \mathbf{x})(k(\mathbf{x}, \mathbf{x}) + \sigma_i^2 \delta_{ij})^{-1} k(x^*, \mathbf{x}) \tag{A.2}$$

$$= k(x^*, x^*) - k(x^*, \mathbf{x}) \left(k(\mathbf{x}, \mathbf{x}) + \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_N^2 \end{bmatrix} \right)^{-1} k(x^*, \mathbf{x}) \tag{A.3}$$

$$= k(x^*, x^*) - k(x^*, \mathbf{x}) V^{-1} k(x^*, \mathbf{x}) \tag{A.4}$$

where the last equation make use of the substitution $V = \left[k(\mathbf{x}, \mathbf{x}) + \sigma_i^2 \delta_{ij} \right]$.

The kernel used is the squared exponential kernel (Gaussian kernel) defined in equation 2.8 as

$$k(x_1, x_2) \;=\; \theta_0^2 e^{-\frac{\|x_1 - x_2\|_2^2}{2\theta_1^2}} \tag{A.5}$$

The derivative of the kernel function with respect to the variable θ_1 is

$$\frac{d}{d\theta_1}k(x_1,x_2) = \theta_0^2 e^{-\frac{\|x_1-x_2\|_2^2}{2\theta_1^2}}\frac{\|x_1-x_2\|_2^2}{\theta_1^3} = k(x_1,x_2)\frac{\|x_1-x_2\|_2^2}{\theta_1^3} \tag{A.6}$$

The gradient of the error function in equation A.1 is used to simplify the expression before evaluating the individual derivatives of the error function. The substitution $e(\phi) = (\sigma^2_{est}(x_k|\phi) - \sigma^2_{obs}(x_k))$ is used to get a more compact notation. The gradient simplification and the two different derivatives are derived in the equations A.7-A.11, A.17-A.22 and A.12-A.16 respectively.

$$\nabla\mathcal{E}(\phi) = \frac{1}{2M}\sum_{k=1}^{M}\nabla(\sigma^2_{est}(x_k|\phi) - \sigma^2_{obs}(x_k))^2 \tag{A.7}$$

$$= \frac{1}{M}\sum_{k=1}^{M}\Big(\sigma^2_{est}(x_k|\phi) - \sigma^2_{obs}(x_k)\Big)\nabla\sigma^2_{est}(x_k|\phi) \tag{A.8}$$

$$= \frac{1}{M}\sum_{k=1}^{M}e(\phi)\nabla\sigma^2_{est}(x_k|\phi) \tag{A.9}$$

$$= \frac{1}{M}\sum_{k=1}^{M}e(\phi)\nabla\Big(k(x_k,x_k) - k(x_k,\mathbf{x})(k(\mathbf{x},\mathbf{x}) + \sigma_n^2\delta_{ij})^{-1}k(x_k,\mathbf{x})^T\Big) \tag{A.10}$$

$$= \frac{1}{M}\sum_{k=1}^{M}e(\phi)\nabla\Big(k(x_k,x_k) - k(x_k,\mathbf{x})V^{-1}k(x_k,\mathbf{x})^T\Big) \tag{A.11}$$

$$\frac{d}{d\sigma_i^2}\mathcal{E}(\phi) = \frac{1}{M}\sum_{k=1}^{M}e(\phi)\frac{d}{d\sigma_i^2}\Big(k(x_k,x_k) - k(x_k,\mathbf{x})V^{-1}k(x_k,\mathbf{x})^T\Big) \tag{A.12}$$

$$= -\frac{1}{M}\sum_{k=1}^{M}e(\phi)\frac{d}{d\sigma_i^2}\Big(k(x_k,\mathbf{x})V^{-1}k(x_k,\mathbf{x})^T\Big) \tag{A.13}$$

$$= -\frac{1}{M}\sum_{k=1}^{M}e(\phi)k(x_k,\mathbf{x})\frac{d}{d\sigma_i^2}\Big(V^{-1}\Big)k(x_k,\mathbf{x}) \tag{A.14}$$

$$= -\frac{1}{M}\sum_{k=1}^{M}e(\phi)k(x_k,\mathbf{x})V^{-1}\frac{d}{d\sigma_i^2}\left(\begin{bmatrix}\sigma_1^2 & 0 & 0\\ 0 & \ddots & 0\\ 0 & 0 & \sigma_M^2\end{bmatrix}\right)V^{-1}k(x_k,\mathbf{x}) \tag{A.15}$$

$$= -\frac{1}{M}\sum_{k=1}^{M}e(\phi)k(x_k,\mathbf{x})V^{-1}\hat{e}_i\hat{e}_i{}^T V^{-1}k(x_k,\mathbf{x}) \tag{A.16}$$

In the step between A.14-A.15 we applied the rule $\frac{d}{dM_{ij}}M^{-1} = M^{-1}\frac{d}{dM_{ij}}(M)M^{-1}$. $\hat{e}_i$ is the identity vector with zeros at all positions except for the $i{-}th$ place where there is a 1. $\hat{e}_i\hat{e}_i^T$ is therefore an N times N matrix filled with zeros except at the

diagonal element (i, i) where there is a 1.

$$\frac{d}{d\theta_1}\mathcal{E}(\phi) = \frac{1}{M}\sum_{k=1}^{M} e(\phi)\frac{d}{d\theta_1}\Big(k(x_k, x_k) - k(x_k, \mathbf{x})V^{-1}k(x_k, \mathbf{x})^T\Big) \tag{A.17}$$

$$= -\frac{1}{M}\sum_{k=1}^{M} e(\phi)\frac{d}{d\theta_1}\Big(k(x_k, \mathbf{x})V^{-1}k(x_k, \mathbf{x})^T\Big) \tag{A.18}$$

$$\begin{aligned} = -\frac{1}{M}\sum_{k=1}^{M} e(\phi)\Bigg(& \frac{d}{d\theta_1}\Big(k(x_k, \mathbf{x})^T\Big)V^{-1}k(x_k, \mathbf{x})^T \\ & + k(x_k, \mathbf{x})\frac{d}{d\theta_1}\Big(V^{-1}\Big)k(x_k, \mathbf{x})^T \\ & + k(x_k, \mathbf{x})V^{-1}\frac{d}{d\theta_1}\Big(k(x_k, \mathbf{x})^T\Big)\Bigg) \end{aligned} \tag{A.19}$$

$$\begin{aligned} = -\frac{1}{M}\sum_{k=1}^{M} e(\phi)\Bigg(& 2\frac{d}{d\theta_1}\Big(k(x_k, \mathbf{x})\Big)V^{-1}k(x_k, \mathbf{x})^T \\ & + k(x_k, \mathbf{x})\frac{d}{d\theta_1}\Big(V^{-1}\Big)k(x_k, \mathbf{x})^T\Bigg) \end{aligned} \tag{A.20}$$

$$\begin{aligned} = -\frac{1}{M}\sum_{k=1}^{M} e(\phi)\Bigg(& 2\frac{d}{d\theta_1}\Big(k(x_k, \mathbf{x})\Big)V^{-1}k(x_k, \mathbf{x})^T \\ & + k(x_k, \mathbf{x})V^{-1}\frac{d}{d\theta_1}\Big(k(\mathbf{x}, \mathbf{x})\Big)V^{-1}k(x_k, \mathbf{x})^T\Bigg) \end{aligned} \tag{A.21}$$

$$\begin{aligned} = -\frac{1}{M}\sum_{k=1}^{M} e(\phi)\Bigg(& 2\frac{dk(x_k, \mathbf{x})}{d\theta_1}V^{-1}k(x_k, \mathbf{x})^T \\ & + k(x_k, \mathbf{x})V^{-1}\frac{dk(\mathbf{x}, \mathbf{x})}{d\theta_1}V^{-1}k(x_k, \mathbf{x})^T\Bigg) \end{aligned} \tag{A.22}$$

The resulting expressions are as follows:

$$\frac{d}{d\theta_1}\mathcal{E}(\phi) = \frac{-1}{M}\sum_{k=1}^{M}\Big(2\frac{dk(x_k, \mathbf{x})}{d\theta_1}V^{-1}k(x_k, \mathbf{x})^T + k(x_k, \mathbf{x})\frac{dV^{-1}}{d\theta_1}k(x_k, \mathbf{x})^T\Big)e(\phi) \tag{A.23}$$

$$\frac{d}{d\sigma_i^2}\mathcal{E}(\phi) = \frac{-1}{M}\sum_{k=1}^{M}\Big(k(x_k, \mathbf{x})V^{-1}\hat{e}_i\hat{e}_i^T V^{-1}k(x_k, \mathbf{x})^T\Big)e(\phi), \qquad i = 1 \ldots N \tag{A.24}$$

They can be evaluated using the derivative of the kernel function found in equation A.6.