

© 2013

Sedat Ozer

ALL RIGHTS RESERVED

ACTIVITY DETECTION IN SCIENTIFIC VISUALIZATION

By **SEDAT OZER**

A Dissertation submitted to the

Graduate School-New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Prof. Deborah Silver

And approved by

New Brunswick, New Jersey

OCTOBER, 2013

ABSTRACT OF THE DISSERTATION

Activity Detection in Scientific Simulations

By SEDAT OZER

Dissertation Director:

Prof. Deborah Silver

Today's state of the art simulations generate high-resolution data at an ever-increasing rate. Such simulations produce data with billions of mesh points (or voxels) for each timestep and thousands of such timesteps with multiple variables. Time-varying data can easily reach peta- and exa-byte scale. Visualizing these massive data sets is still an on-going problem. Even after visualizing this data, viewing each variable at each timestep is practically impossible when there are thousands of timesteps. Simulations become too complex for the scientist to analyze manually. In such time-varying data sets, scientists want to know "where and when events happen" or "how long an event lasts". Finding these events in thousands of timesteps is not possible with

standard visualization tools. What scientists need are routines, procedures and visualizing techniques to help filter massive data and help focus on areas and events of interest automatically.

The problems facing any attempt to localize complex events (activities) automatically in time-varying 3D scientific data can be summarized as: (1) provide an appropriate way for users to define an event of interest; (2) find an appropriate formalism to model this event; (3) apply the model to detect many instances of the event of interest in simulation data; and (4) present the detected events to users in an appropriate visual form.

The contributions in this dissertation include introduction of the concept of activity detection for scientific visualization, the use of Petri Nets to model and detect activities in scientific visualization, an enhancement of Petri Nets to include the dynamics of scientific phenomena and demonstration of the use of activity detection on three different 3D time-varying data sets as case studies. In addition, a full 3D group-tracking model in which we extract and track groups as well as the individual features that form them is presented.

TABLE of CONTENTS:

ABSTRACT OF THE DISSERTATION.....	ii
LIST OF FIGURES.....	vi
CHAPTER 1: INTRODUCTION	1
1.1. Motivation, problem statement	4
1.2. What is activity detection in scientific visualization?	6
1.3. Enhancing scientific visualization	11
1.4. Summary of contributions.....	13
CHAPTER 2: THE PROPOSED ACTIVITY DETECTION FRAMEWORK.....	16
2.1. Definitions	16
2.2. An overview of the proposed activity detection framework	21
2.2.1 Segmentation.....	24
2.2.2 Tracking	27
2.2.3 Activity detection	31
CHAPTER 3: BACKGROUND – RELATED WORK	34
3.1. Related work in computer vision	34
3.2. Related work in visualization.....	39
3.3. Activities as patterns and the use of machine learning	40
CHAPTER 4: GROUP TRACKING.....	42
4.1. Overview of the group tracking algorithm	44
4.2. Group extraction via clustering.....	48
4.3. Similarity functions	50
4.4. Group tracking & group events.....	51
4.5. Group & cross-level events	55
4.6. Creating a domain specific similarity function.....	59
4.7. Results	62
CHAPTER 5: PETRI NETS FOR ACTIVITY DETECTION	72
5.1. Fundamental Petri Net concepts and components.....	73
5.2. Marked Petri Nets	75
5.3. Timed and stochastic Petri Nets.....	77
5.4. Coloured Petri Nets.....	79
5.5. Earlier definitions of the enabling and firing processes	79
5.6. Unaddressed problems for activity detection.....	81
CHAPTER 6: TOKEN-TRACKING PETRI NETS FOR ACTIVITY DETECTION	84
6.1. Activity detection framework with TTPN	90

6.2. Modeling with Petri Nets	91
CHAPTER 7: ACTIVITY VISUALIZATION	95
CHAPTER 8: IMPLEMENTATION.....	100
8.1. Implementation of feature & group tracking	100
8.2. Implementation of TTPN	103
CHAPTER 9: EXPERIMENTAL RESULTS FOR ACTIVITY DETECTION	109
9.1. Merge-split activity detection in turbulent vortex structures	109
9.2. Detecting anomalous bending in acoustic plume scans.....	115
9.3. Packet formation in wall bounded turbulence flow simulations.....	120
CHAPTER 10: DISCUSSION AND CONCLUSION.....	124
ACKNOWLEDGEMENTS, PUBLICATIONS AND PRESENTATIONS.....	128
REFERENCES.....	130
APPENDIX – I: ATTRIBUTE GENERATION.....	136
A1.1. Scale-invariance feature transform (SIFT).....	139
A1.2. Scale space extrema detection.....	140
A1.3. Keypoint localization	147
A1.4. Orientation assignment.....	149
A1.5. Keypoint descriptors	150
A1.6. 3D-SIFT and SIFT-like algorithms	152
APPENDIX – II: USER MANUAL	153
APPENDIX – III: DATA SETS	168
A3.1. Pseudo-spectral simulation of coherent turbulent vortex structures .	168
A3.2. Acoustic plume scans	169
A3.3. Wall bounded turbulence flow simulations	170

LIST of FIGURES:

Figure 1: Inputs and outputs of a typical activity detection framework.....	3
Figure 2: Scientific Data examples	5
Figure 3: Iso-surface visualization of vortices in a time-varying 3D computational fluid dynamics simulation.....	6
Figure 4: A simple sequence of an activity	7
Figure 5: Flow diagram of hypothesis testing.	12
Figure 6: A relational diagram illustrating the flow in scientific visualization	20
Figure 7: An illustration of the extracted and tracked features in the data.	22
Figure 8: An Activity Detection Framework with Petri Nets.....	23
Figure 9: A generic activity detection framework	32
Figure 10: A classification schema for various activity detection techniques	38
Figure 11: An example of a hierarchical structure	43
Figure 12: Group tracking framework flow diagram	46
Figure 13: An illustration of volume-overlap based group tracking	54
Figure 14: Single Feature Track events vs. group tracking events.....	56
Figure 15: An illustration of full merge, partial merge, full split and partial split ..	57
Figure 16: An illustration of cross-group event	59
Figure 17: Visualization of feature tracking vs. group tracking in wall bounded turbulent flow simulation data	64
Figure 18: The evolution of a selected feature and its group is visualized in the first five timesteps.....	66
Figure 19: The total number of extracted features vs. the total number of identified packets in each timestep.....	67
Figure 20: Group tracking results are visualized in timesteps 1, 2 and in timestep 8	69
Figure 21: Group tracking results are visualized in timesteps 21, 26 and in timestep 31.....	70
Figure 22: Group extraction is shown on the large data set	71
Figure 23: Illustration of Petri Net components on the Petri Net model of the unattended bag example.	75
Figure 24: An example of enabling and firing a transition in a marked Petri Net.	78
Figure 25: Various illustrations of different time dependent problems in Petri Nets.	87
Figure 26: The use of sub-nets in TTPN.....	89
Figure 27: Flow diagram of the proposed activity detection framework.....	92
Figure 28: Illustrative examples of (a) conflict and (b) deadlock.....	93
Figure 29: An example of graph based activity visualization	96
Figure 30: Illustrative example of activity-histogram	97
Figure 31: An example of isolated activity visualization.....	99

Figure 32: The modules of the feature and group tracking implementation	100
Figure 33: The modules included in the Petri Net implementation are shown ..	105
Figure 34: The Petri Net data structure used in TTPN.....	108
Figure 35: The Petri Net model of the merge-split activity	109
Figure 36: The total number of detected activities changes, as the value of k_0 changes.....	111
Figure 37: Three detected instances of the “merge-split” activity are visualized	112
Figure 38: Two other detected instances of the “merge-split” activity are visualized.....	113
Figure 39: Another combination of merge, split and continue events is modeled	114
Figure 40: A Petri Net model for “Anomalous Plume bending” is shown	117
Figure 41: “Anomalous Plume bending” detection in a time-varying 3D plume data set is visualized	118
Figure 42: The Petri Net model for “Anomalous Plume bending” is shown	119
Figure 43: Various information of the wall bounded turbulence DNS	122
Figure 44: Sample detected packet formations are visualized in wall bounded turbulence DNS	123
Figure 45: An illustration of two different objects with the same mean and variance values.....	137
Figure 46: Illustration of the concept scale in scale space.....	140
Figure 47: Computing the local extrema points.	143
Figure 48: Visualizing the first and second octaves of a given image.....	144
Figure 49: The effect of scaling at various octaves and scales.....	145
Figure 50: Computed DoG images for Figure 49.....	146
Figure 51: Keypoint descriptor computation process.....	151
Figure 52: A sample configuration file for feature and group tracking module..	156
Figure 53: A sample configuration file for the Petri Net module.....	159
Figure 54: A sample from the content of an *.attr file.....	162
Figure 55: A sample from the content of an *.group file.....	165
Figure 56: A sample from the content of an *.trak file.....	166
Figure 57: A schematic diagram of acoustic plume scan process	170
Figure 58: in wall bounded turbulence simulations	171

CHAPTER 1:

INTRODUCTION

Today's state of the art simulations and data acquisition systems generate high resolution data at an ever increasing rate. These simulations are generally in 3D with many variables and many timesteps. They produce data with billions of mesh points (or voxels) for each timestep and thousands of such timesteps with multiple variables. Time-varying data can easily reach peta- and exa-byte scale. Visualizing these massive data sets is still an on-going problem. Even after visualizing this data, viewing each variable or each object "feature" at each timestep is practically impossible in thousands of timesteps. Simulations become too complex for the scientist to analyse manually for simulations with identifiable features. Scientists want to know "*where and when events happen*" or "*how long an event lasts*". Finding these events in thousands of timesteps is not possible with today's tools. What scientists need are routines, procedures and visualizing techniques to help filter massive data and help focus on areas and events of interest. Furthermore, in many simulations, scientists have hypothesis about events occurring in the data and would like to test and refine their assumption. Allowing a scientist to model an event and then search for that event over thousands of timesteps would both filter massive data and enable scientists to focus on regions of interests in space and time.

Detection of events has been an active research area in video analysis and there have been a large number of techniques and tools have been proposed

(see Chapter 3). However, currently there is no tool available for scientists to define, model and automatically search for complex events, i.e., activities, in their time-varying 3D scientific data. Most visualization and analysis routines are still focused on a single timestep. Available visualization routines for time-varying data are mostly concerned with the correspondence problem which involves correlating objects from one timestep to the next. However, these routines do not provide the scientist with the ability to model complex spatio-temporal patterns or to answer the fundamental issue of where, when and how “interesting things” occur.

Activity detection is an automated search process for finding a specific and complex pattern (activity) in a large data set containing many different types of patterns. Activity examples include formation of features (such as galaxies, halos, storms or blood clots), anomalous interaction or behaviour (anomaly detection), merge-split or ignition events. These events are distributed over a large number of timesteps. Different portions of the pattern happen at different timesteps. The duration of an activity changes from one instance to another, i.e., one instance of an activity may take 20 timesteps and another instance of the same activity could take four. Inputs and outputs of an activity detection framework are shown in Figure 1. Inputs to a typical activity detection system are the data set and the definition of the activity. The output of an activity detection framework is the list of detected activities including the answers to the “where, when and which” questions.

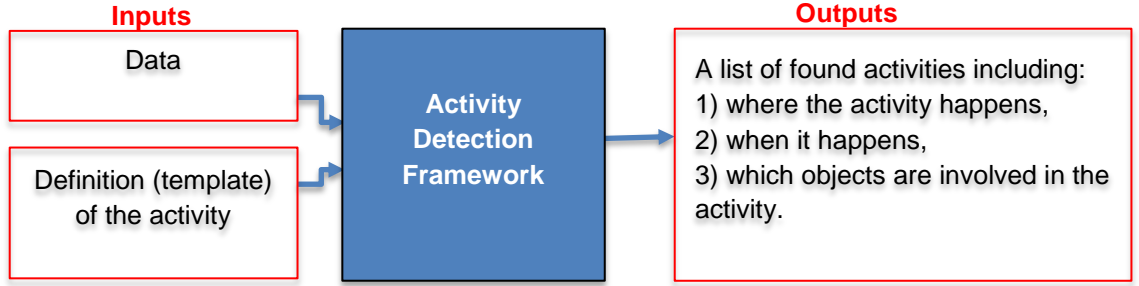


Figure 1: Inputs and outputs of a typical activity detection framework. In addition to the typical data, a form of the activity definition is also given to the framework. This form may be in the same format as the input data, in a form of a mathematical equation or can be a semantic description, etc. The output of the framework is the list of detected activities.

The problems facing any attempt to localize complex events (activities) automatically in time-varying 3D scientific data can be summarized as follows: (1) define an event of interest; (2) model this event; (3) detect all the instances of the model in simulation data; (4) visualize the results.

In this dissertation, we introduce activity detection and discuss the applicability of activity detection for both data analysis and visualization purposes. ***The main goal is to develop a framework that a scientist can use to first model a spatio-temporal pattern and then search through massive data sets to find instances of such a pattern.*** The natural way of modeling events or activities is using a graphical and state based approach that can convert or translate the semantics of an event into a graph-based sequential model. Therefore, in this dissertation, Petri Nets are proposed to model and search events in scientific simulations. The focus primarily is the activities of observable features. However, the presented techniques can be used for detecting any type of activity.

1.1. Motivation, problem statement

Visualization routines for time-varying 3D data are heavily focused on the problems of feature tracking [71]. *Feature tracking* correlates the features (scientific objects) from one timestep to the next in time-varying scientific simulations. Examples of scientific data sets are shown in both Figure 2 and Figure 3. Figure 2a visualizes the first timestep of an ocean simulation where the region of interests are the ocean eddies and Figure 2b visualizes the first timestep of a 3D computational fluid dynamics simulation where the region of interests are the vortices. Figure 3 visualizes sample timesteps from the processed data (extracted and tracked vortices) by applying the feature tracking algorithm on the entire fluid dynamics simulation data (the data set has 100 timesteps). While the available feature tracking techniques allow time-varying data analysis in 3D simulations, they do not provide the scientist flexibility to model different activities or hypothesis about the data to answer the fundamental issue of *where and when* “interesting things” occur.

Scientists are usually interested in analysing specific patterns and studying local interactions, the origin of features, how they evolve, and how they interact. These patterns can span multiple timesteps and can occur frequently throughout the simulations. Many of the patterns include different types of feature states or their various types of interactions. In this dissertation, we call these time dependent patterns “events” or “activities”.

Relating to Figure 3 and the vortex simulation dataset, sample questions a scientist may ask of the data include:

- Which features first merge and then split within a period of five timesteps (merge-split activity)?
- How many features do further split among the features that already performed the “merge-split” activity defined above?
- How long does it take for features to merge-split-split and then merge again?
- Which features rotate around other objects and where do these events occur?
- When does a feature type transform into another feature type and how long do these transformation events take?

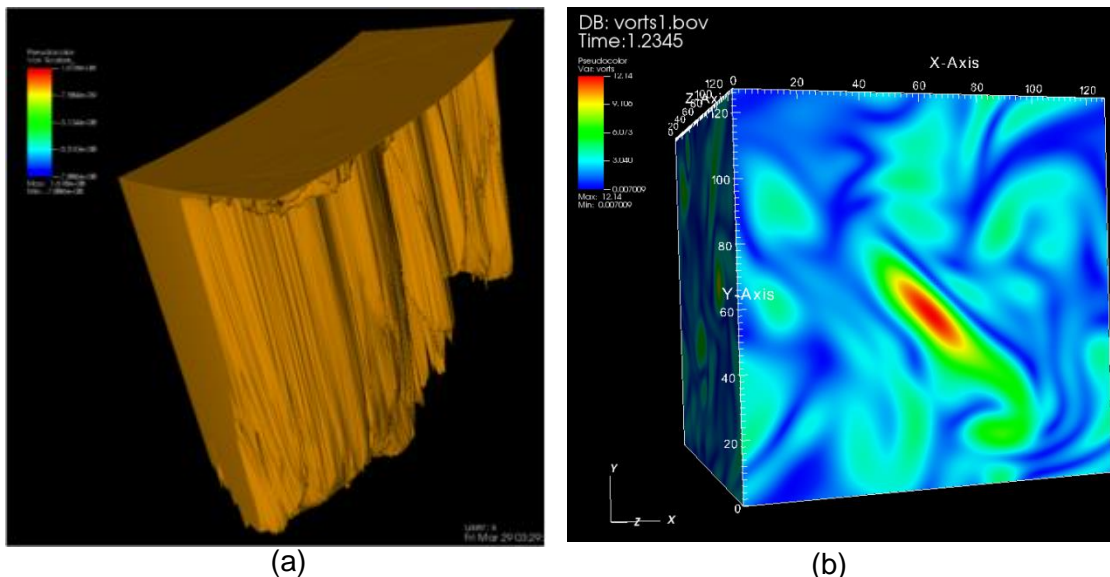


Figure 2: Scientific Data examples: (a) the visualization of the first timestep of an ocean simulation, (b) visualization of the first timestep of a computational fluid dynamics simulation.

Answering similar questions in the computer vision community has been an active research area and is referred to as “activity detection” or “activity

recognition”. However, this is not an active topic of research in the 3D visualization community.

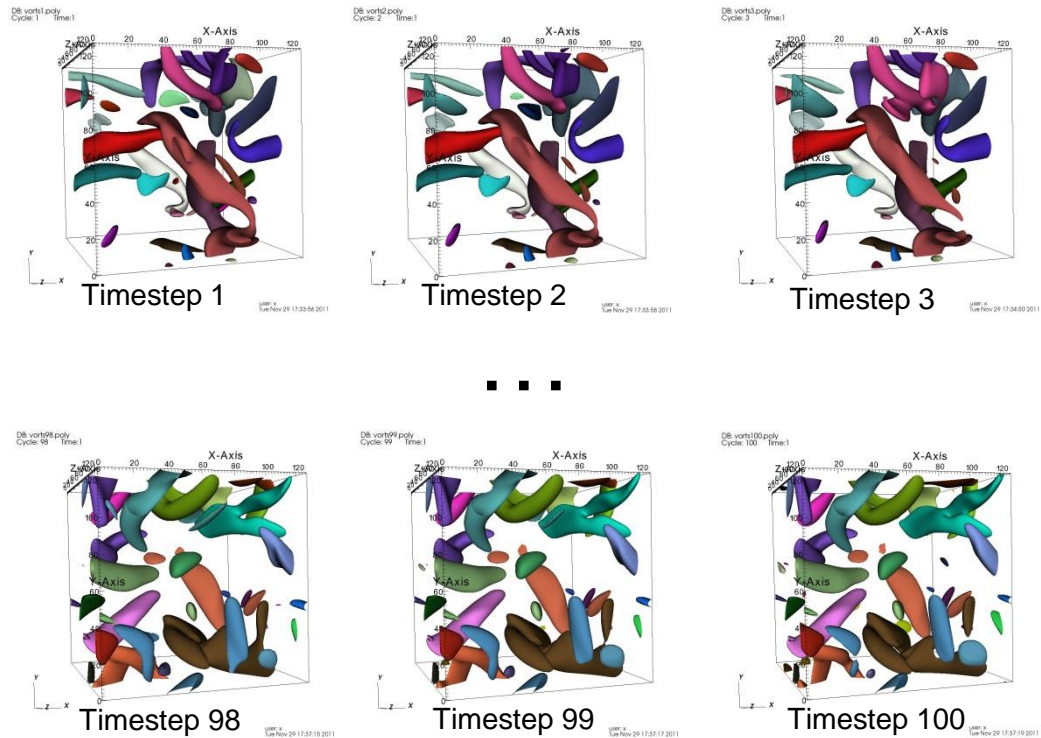


Figure 3: Iso-surface visualization of vortices in a time-varying 3D computational fluid dynamics simulation. The extracted and tracked vortices in the first three timesteps and the last three timesteps of the simulation data are shown above. Each individual vortex has a unique color. The splitting vortices have the same parent vortex’s color. The data set is from [71].

1.2. What is activity detection in scientific visualization?

Activity detection is an active research field in video analysis. For example consider a security surveillance system at an airport. There are hundreds of locations and thousands of hours of video that must be monitored. One situation of interest to security personal is a “leaving a bag unattended” activity where a person walks in with a bag, puts the bag down and then walks away without the

bag. In Figure 4, a model of this activity is shown as a sequence of key and atomic (primitive) events. This sequence is shown as a directed graph. Clearly, we can add complexity to this sequence, for example, by adding more than one person where there is a group of people holding a bag or where the person routinely picks up a bag and puts it down. In a video sequence, each step would occur at a specific point in time. An example of an activity sequence from a video is given in Figure 4. Underneath each activity state, a sample time step is given representing where in the dataset that event is detected. In another video sequence, the detection would occur in different time steps and the time difference between the states would vary.

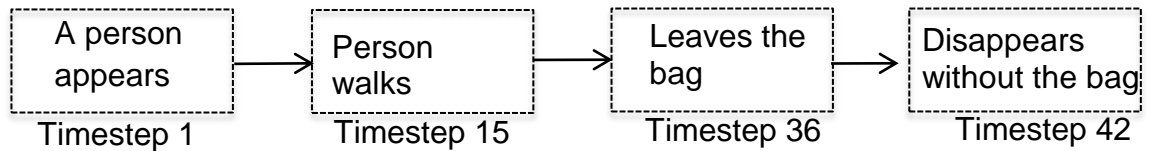


Figure 4: A simple sequence of an activity. The activity is formed of four different atomic events (actions) over the course of 42 timesteps.

Many other activity examples have been searched in video data such as:

- a person leaves his/her car to meet another person in another car as in [25],
- a person robs a bank as in [3],
- a person skips the checkpoint at an airport as in [41],

All these activities involve actors performing the activity, contain a certain semantic description and span over multiple timesteps. Furthermore, they can all be categorized by a sequence of timesteps.

Similar to Figure 4, there are many cases where the scientists are looking for complex interactions of features in 3D scientific simulations. Extraction of such events and their quantification is important for data analysis in scientific data sets.

While detecting particular activities has been an active research area in video analysis, activity detection remains an open research problem in 3D scientific visualization. The problems facing any attempt to localize activities automatically in time-varying 3D scientific data are four-fold, namely,

1. How to define and model a complex interaction that spans a limited time,
2. How to detect simultaneous activities efficiently when there are multiple actors performing the same activity independently in the scene,
3. How to use an activity model to automate and ease the querying and visualization processes,
4. How to develop a generic method that can be used in many different domains in scientific visualization, i.e., a method that does not require adapting the available source code to each domain or to each application.

The first problem is related to the inference problem in computer vision and has been an active research area under the “activity detection” name, (although the terms “activity recognition”, “action detection” and “action recognition” have also been used interchangeably). In this dissertation, we propose to use Petri Nets to model an activity in scientific visualization.

The second problem is the detection of multiple activities simultaneously where different actors are performing the same activity independently. When

there are multiple objects in the scene, each object is likely to perform the same activity independent of the other possible activity performers (objects). Although there have been recent studies in video, surveillance and multimedia fields to detect activities with Petri Nets, the main goal has been the detection of a single activity performed by one or many people or a person's interaction with the surrounding objects (mainly other humans or vehicles). In many scientific simulations the number of objects is greater than one, thereby increasing the likelihood of observing more than one activity happening in a scene over time. This problem is a common problem in both computer vision and scientific visualization communities. To handle this case, we enhance Petri Nets to detect simultaneous activities of scientific objects.

The third problem in attempting to automatically localize events is related to the efficient integration of the activity detection with scientific visualization. Activity detection archives a list of participating features, their locations and associated timesteps. This information can be used to enhance scientific visualization by isolating features in each timestep or even designing observable feature based time-varying transfer functions.

Any specific activity can always be implemented directly for a simulation. However, this approach is not scalable and does not easily allow scientists to re-model or reuse these activity programs. With Petri Nets, we provide scientists a generic method to model and detect events in time-varying 3D data.

In this dissertation, we present 3 different examples. The first data set is a toy data set. It contains a pseudo-spectral simulation of coherent turbulent vortex

structures. In this data set, features merge, split throughout the data set and we look for the features that perform a specific combination these events. The combination we seek is “merge-split” activity. The second data set is a collection of acoustic scans of a plume in the sea over time. In this data set, the scientists are interested in detecting the specific timesteps when unusual changes in direction or magnitude of the bending of the plume in response to local ocean currents occur. The third data set is a wall bounded turbulent Direct Numerical Simulation (DNS). In this data set, there are hundreds of hairpin vortices (features) interacting with each other in each timestep. One event, which interests scientists, in such simulations is finding when several “young” hairpins come together and eventually form a group of unconnected hairpins moving together. These are the three examples we have chosen to explore. However, in many other domains, there are problems with similar need for methods of detection of events such as blood clot formation in blood flow simulations, extinction and re-ignition in turbulent flames in combustion simulations or magnetic storm formations in space weather simulations.

These abovementioned scientific objects mimic the computer vision example in which actors perform activities: the actors are now features in the time-varying scientific data and the activities reflect interactions or changes in feature states. The ability to model and detect such activities where and when they take place would be helpful in these situations.

1.3. Enhancing scientific visualization

Activity detection can be used as an analysis and visualization tool. These ideas are summarized below:

- **Activity detection to model or validate an hypothesis in scientific simulations:** Many scientific simulations are generated to simulate a specific event or phenomena. Such phenomena are modeled by a set of hypothesis. Activity detection can be used to detect and thus validate, such phenomena happening within the data. For example, if there is no such event found in the data, then the original hypothesis may not be correct.

While scientific simulations are the alternatives to the experimental setups for data gathering, there are specific issues in simulations. One of the fundamental problems in simulations, (as opposed to the physical and economical limitations in experimental studies) is establishing a successful model describing the phenomena. The model in many cases is not clearly known in advance, and therefore it usually requires several iterations to refine. The refinement process is done via an interactive and recursive process, in which first the simulation data is generated based on the model and then the generated simulation data is analysed, and then the model is tuned or restructured based on the analysis results. Activity detection can be used to help in hypothesis generation and validation.

Figure 5 shows the flow diagram for hypothesis testing. In scientific simulations, the scientist usually has an idea (hypothesis) about possible phenomena happening in the data and would like to analyse and check those phenomena. For that purpose, first the scientist creates the simulation data, extracts and tracks all the region of interests in the simulation and computes their attributes. These computed attributes help scientist to model his/her hypothesis first, and then the activity detection algorithm would extract all the instances of the model. The visual results would provide feedback to scientist to refine their model (i.e., hypothesis). The red loop (shown in Figure 5) would repeat itself for model refinement.

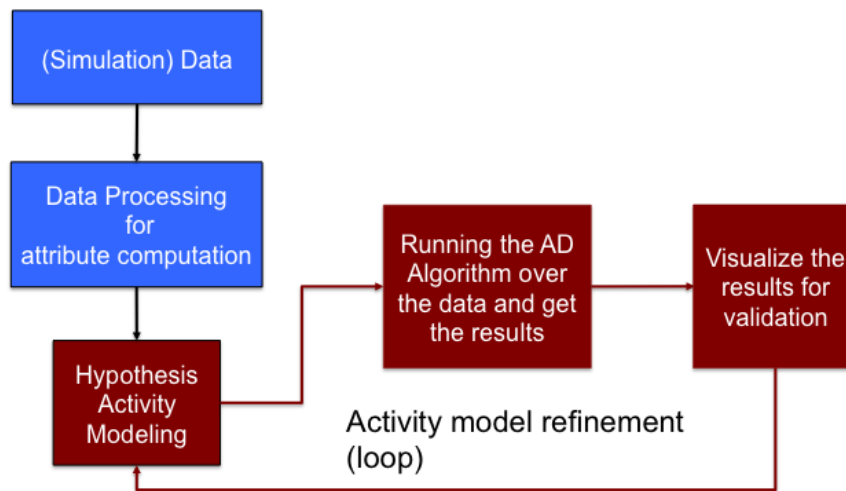


Figure 5: Flow diagram of hypothesis testing. Once the attributes are computed, scientist can go through the loop to refine his/her hypothesis.

- **Activity detection for time-varying data analysis:** Once the scientist is assured that the hypothesis is correct, then he/she can search for various types of events within the data. Searching for various types of events, the number of such events occurring and the duration of such events can help

the scientist in understanding their data and ultimately the phenomena of interest. Specifically, the statistics of the participated features, the numbers of partially completed activities and their percentage can be used to analyze the scientific data and scientific phenomena.

- **Activity detection for improving the visualization of time-varying data:** Activity detection can help reduce clutter, and can focus on events rather than focusing on features. This is especially useful in data abstraction for visualization. While not the focus of this dissertation, activity detection can also help model and generate automated time-varying transfer functions. Please see Chapter 7 for more about activity visualization.

1.4. Summary of contributions

The contributions of this dissertation are listed as follows:

- 1) Introduction of the concept “activity detection” for scientific visualization,
- 2) proposal of the use of Petri Nets to model and detect activities in scientific simulations,
- 3) enhancement of the existing Petri Net formalism to handle simultaneous activity detection in time-varying 3D data with the introduction of token-tracking Petri Nets,
- 4) the use of a similarity based clustering algorithm to first define and then group features automatically in scientific datasets,
- 5) proposal of an algorithm for tracking groups (besides the features) in scientific datasets,

- 6) description of how activity detection can be used to enhance scientific visualization, and
- 7) demonstration of the use of activity detection on three different 3D time-varying data sets.

In Chapter 2, first a list of the descriptions of the commonly used terms in this dissertation is given and then an overview of the proposed activity detection framework and its parts including segmentation, tracking and activity detection are discussed. The related literature in activity detection is provided in Chapter 3. Chapter 4 presents an algorithm that enhances the feature tracking algorithm to group the features and track these groups as well as the features.

Chapter 5 provides the background for Petri Nets to model and detect activities with Petri Nets. Chapter 6 presents the enhancements in Petri Nets that were made as a part of this dissertation. These enhancements consider the dynamics of scientific simulations. Resulting Petri Net is called token-tracking Petri Nets and can be used for both activity modelling and detection in scientific visualization. Activity detection in scientific simulations also enhances visualization. Such activity visualization techniques are discussed in Chapter 7. Chapter 8 presents the details of the implementations of the proposed activity detection framework. Three case studies of activity detection in scientific visualization are provided in Chapter 9. In Chapter 10 we conclude this dissertation.

Appendices include additional information that complements this dissertation. Appendix I discusses the importance of attribute computation by giving the

details of a specific attribute computation technique (SIFT). Appendix II includes the details of the format of both inputs and outputs in the presented activity detection framework. Appendix III provides background on the data that is used as the input for the proposed activity detection framework.

CHAPTER 2:

THE PROPOSED ACTIVITY DETECTION FRAMEWORK

In this chapter, we provide the definitions and background needed to understand the project and its context. We start with a list of common terms and how we intend them to be understood, then introduce the activity detection framework, and finally review various aspects of activity detection.

2.1. Definitions

Below is a set of commonly used terms.

Data: In this dissertation, we will use the term *data* to describe a set of numbers (values) representing the entire set of measurements in a domain. The values at each node can be single (scalar) or can be vector (multiple values for each node). If these measurements come from different locations of a geometric space, then we will call the data *spatial* data. Therefore spatial data, besides the measurements (data values), also includes the location of these data values and typically includes the connectivity information.

Region of interest (ROI): A region of interest (ROI) is the spatial area or the volume that covers a certain *set of interest points* (i.e., nodes) in the data. The definition of the “interest” depends on the application domain and in this dissertation we restrict it to a scientific object in an observable form. Therefore, in the data, all the nodes that are not a part of any region of interests form a new type of object: *background*. Generally speaking, the term *background* is usually

used to describe the unimportant part of the data and *region of interest* is used to describe the important part of the data.

Object, feature: We will use the terms *object* and *feature* interchangeably referring to a region of interest that has a physical meaning or that is in an observable form. An object or a feature can be a connected set of regions or a group of unconnected regions. For example a human can be an object in one context in a video data, while in another data set a cluster (group) of humans can be defined as a single object.

Group: A group is a set of related objects. The relation can be only logical (such as all the green objects, the set of voxels with a certain value, etc.) or can also have a physical meaning (such as a flock of bird, a school of fish, a galaxy, etc.).

Meta-data, attributes: We will use the terms *meta-data* or *attributes* to describe a set of computed quantitative properties of the objects that are usually obtained by processing the *data*. Examples include the mean value of an object's member points, volume, centroid or surface points of an object.

Spatio-temporal Pattern: A spatial pattern is a pattern that can be defined by only the geometric (or spatial) properties of the data such as shape or length. Spatio-temporal pattern is a pattern that is a function of both space and time. That is, the pattern does not occur in a single timestep. Instead, it spans over multiple timesteps to occur. Detection (or recognition) of such patterns, typically require inspecting sets of timesteps. Therefore spatio-temporal patterns cannot

be detected by inspecting only the geometric structures in a single (each) timestep.

Event: An event can be defined in many different ways including: 1) the cause of the change in an object's state, 2) a spatio-temporal pattern, or 3) an entire eco-system including the cause, objects and the change in object states over the time. System approach is more generic and it allows us to model and analyse an event by using the available system based techniques and tools (such as Petri Nets). In such systems, an event can be considered as a spatio-temporal pattern and can be modelled by considering various causes that result a change in objects' state. An event happens over a course of timesteps and can include the interactions of different types of objects. Events can be further divided into two groups: atomic events (actions) and complex events (activities).

Action: An *atomic event* or *action* is the primitive event that happens in a short time span. Actions usually occur between two consecutive timesteps. For example, an object may split from one timestep to the next [71]. Therefore, in many cases, an action can be inferred by comparing the current timestep to a specific (reference) timestep.

Activity: A complex event or an *activity* is a set of actions that spans multiple timesteps and can include multiple object types, object states or object interactions. Complex events, i.e., activities, usually occur over more than 2 timesteps.

Actor: An actor is an object that participates or is involved in an activity. We will use the terms actor, object or feature interchangeably to refer to a region of interest that performs the activity or is participating in the activity.

Phenomenon: In this dissertation, we will use the term phenomenon specifically referring to a scientific spatio-temporal pattern.

Segmentation, extraction: Segmentation (or extraction) is the process of defining a set of voxels as a region of interest. During segmentation, each voxel in a 3D data is labelled as part of a ROI or as part of the background.

Object recognition: Recognition is the process of assigning a label to each segmented data point (node) from a given set of labels (such as flame, jet, bubble, vortex, hairpin, etc.). Object recognition is the process of specifying the labels for the regions of interests. In scientific simulations, usually a set of connected points (nodes) represent a single (scientifically meaningful) object, therefore such set of connected data points collectively are assigned the same label or ID.

Scientific visualization: Scientists study scientific phenomena through experiments to analyse and understand the underlying dynamics. These experiments yield large amounts of data. Data collection through physical experiments (setups) can be expensive in some domains. In others, it may be almost impossible due to the physical limitations. This is where the scientific simulations become useful and feasible to understand the scientific phenomena. Figure 6 illustrates the data collection and visualization schema in scientific domains.

Scientific data can be either simulation based or real (collected by means of various data acquisition tools). In this dissertation, we will use the both data types (i.e., simulation and real data). The type, the format and the properties of scientific data changes from field to field depending on the properties of the domain such as the topology of the domain, its shape or data dimensions. In this dissertation, we focus on 3-Dimensional (3D) time-varying data sets in which the data is collected and saved in 3 dimensional space (i.e., in a volume) over a course of timesteps. Each 3D volume consists of nodes (voxels).

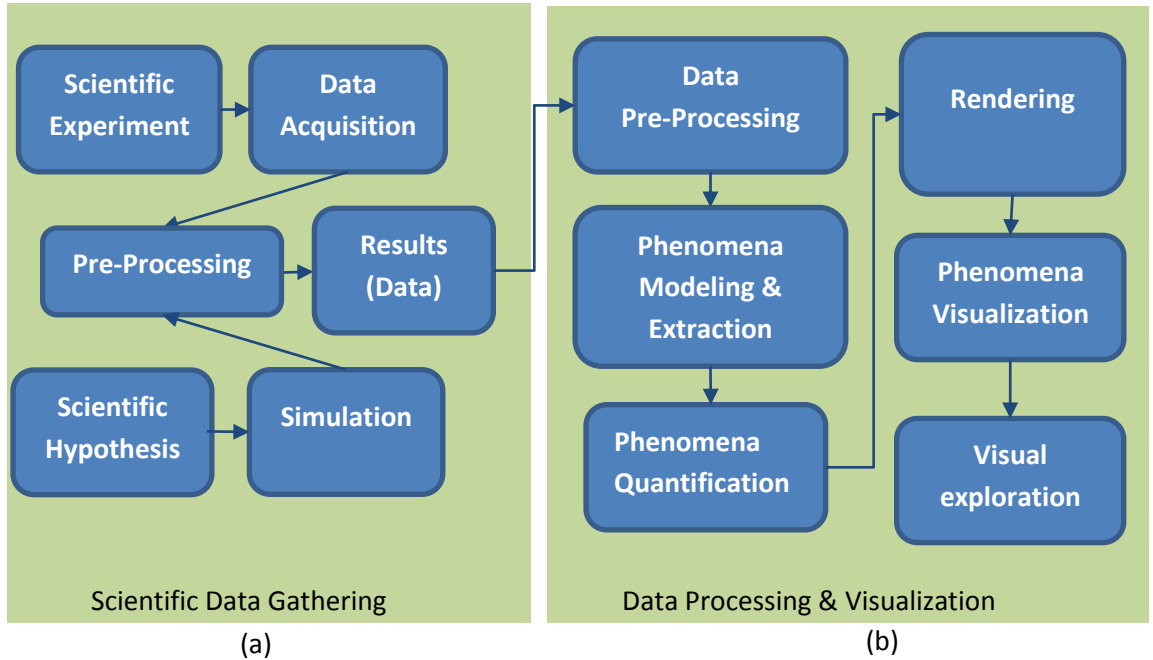


Figure 6: A relational diagram illustrating the flow in scientific visualization. Two major parts of scientific visualization are separated: (a) scientific data collection and (b) data processing & visualization.

Each node can have a scalar or a vector measurements (or values); and each of these values is a function of time, i.e., these values may change over time. Moreover, in each 3D volume, the connectivity information of the nodes is known (given). A time-varying 2D data set can also be considered a 3D data set

by considering the 2D data as a layer (a slice) and then by adding another layer (where all the values are either equal to the minimum or maximum value of the original 2D data) on top of the original 2D data.

2.2. An overview of the proposed activity detection framework

A typical activity detection framework involves actors and the actors are separated from the background. In 3D scientific simulations, this process involves segmentation and tracking.

Since an event, fundamentally, is the cause or the result of a change in an actor's state, it is essential to extract the object (actor) first and then define its state in the data. This information can be computed after the segmentation step. Since an activity spans over multiple timesteps, it is also essential to track the objects over time. Therefore, before the activity detection step, one needs to segment and track the actors. The effect of segmentation (extraction) and tracking in visualization are illustrated in Figure 7. Segmentation (extraction) quantifies the features in the data. Thus only the voxels that are part of a feature can be visualized in the data. Tracking correlated the features from one timestep to the next and provides the correspondence information. By using the correspondence information we can assign the same color to the same segmented object (even if it moves or changes its shape) over time.

Figure 8 shows the overall framework for activity detection with Petri Nets in scientific visualization. The input to the system is the time-varying data set and the Petri Net model defined by the scientist. The first step is processing the data in the framework. In this step, features, groups, variable changes or other types

of user interested entities are computed. The computed meta-data includes spatial properties of the objects and may include volume, mass, centroid, max and min locations, max and min positions, orientation, shape information, bounding box, etc.

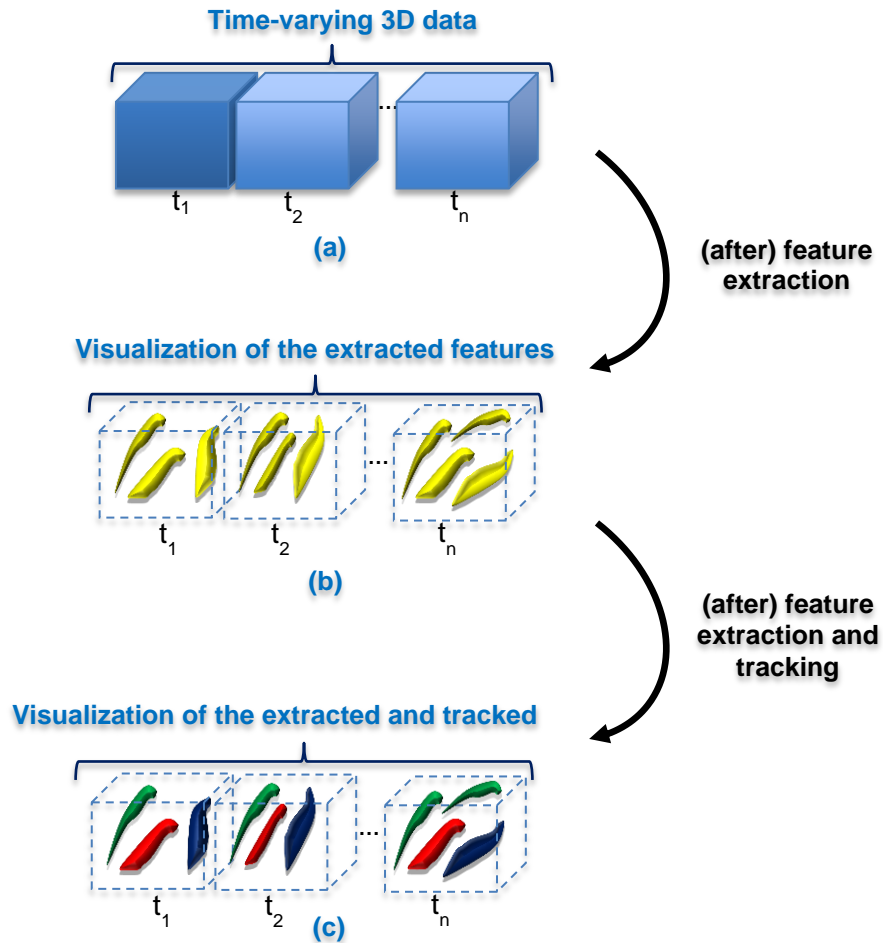


Figure 7: An illustration of (a) time-varying data, (b) visualization of the extracted features in the data, (c) visualization of the extracted and tracked features in the data.

The next step in the framework is correlating the meta-data over time. Correlating the objects (thus their spatial attributes) is known as the correspondence problem.

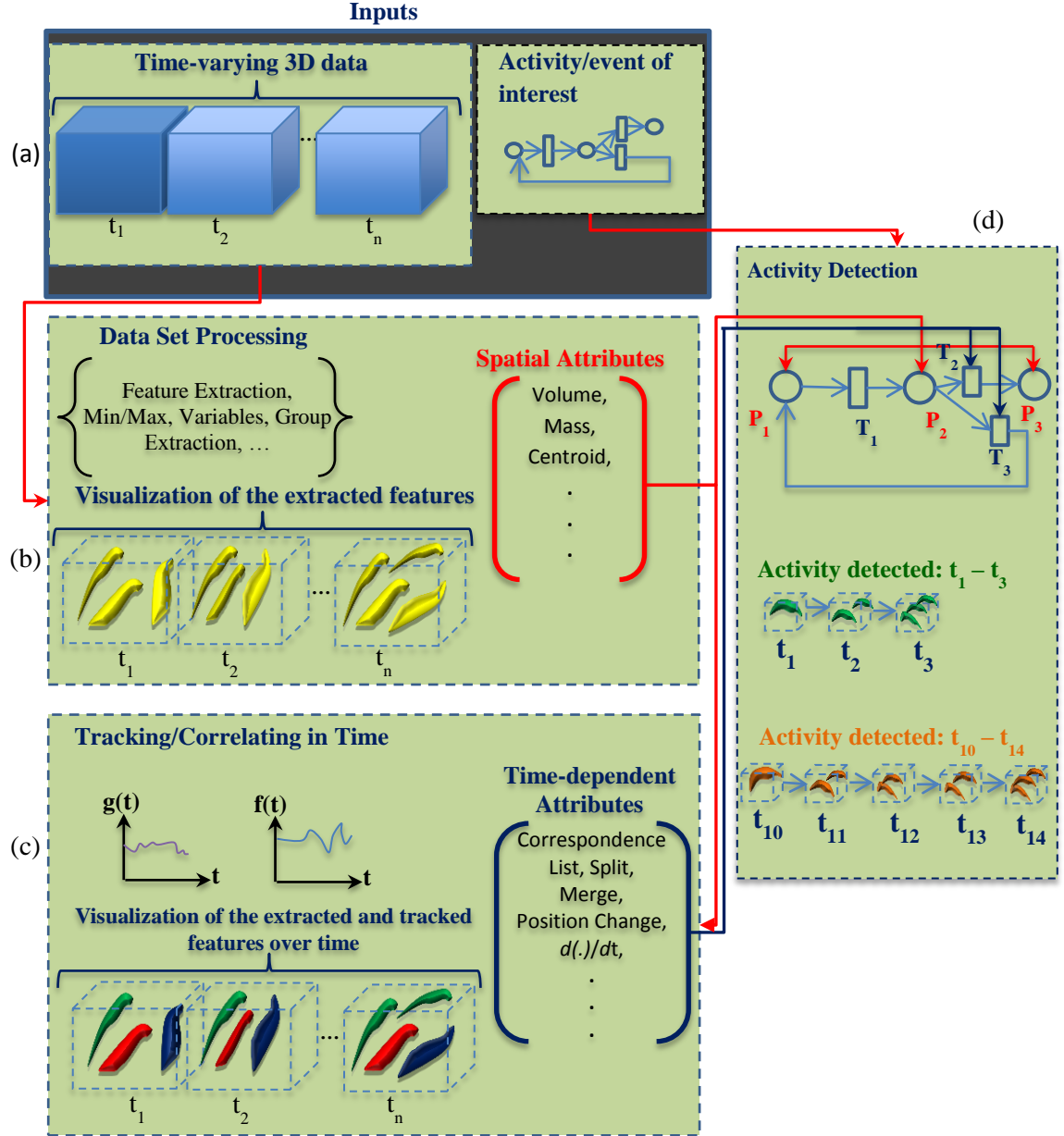


Figure 8: An Activity Detection Framework with Petri Nets. (a) the inputs are the data and the activity model. The activity or event of interest is defined by the scientist and it is the event that the scientist is interesting in searching for within the dataset. The way the scientist describes the event is through the use of a Petri Net. (b) the next step is processing the data. In this step features, groups, etc. are computed and stored as meta-data. (c) This meta-data is used to correlate features/variables and their changes over time. These time-dependent attributes are added to the meta-data. (d) once this information (meta-data) is computed, it can be searched using the Petri Nets to find an activity of interest. The output of the Petri Net is a list of timesteps where the event occurs and a list of features that participate in the activity.

To solve this problem one of many available tracking algorithms using volume overlap, location estimation or shape matching can be used to correlate these objects; or the tracking information may be inherent in the simulation. Time-dependent attributes may include tracking history of the features (the correspondence list), position changes, and any other value/attribute that is a function of time (including time dependent derivatives). All these computed meta-data (both the spatial attributes and the time dependent attributes) is fed into the final step for activity detection. Activity detection uses this meta-data along with the activity model as the inputs to search for the specified activity.

The final step is searching throughout the input meta-data for the activity. The Petri Net is evaluated (run) over time by means of tokens (described in Chapter 5). Tokens represent features undergoing activities and are updated as they move through the Petri Net. The tokens that complete the sequence are the ones that perform the activity.

As a summary, a typical activity detection framework includes three fundamental processes: 1) segmentation and attribute computation, 2) tracking and 3) activity detection.

2.2.1 Segmentation

Segmentation, i.e., *extraction* or *background removal*, is the process of labelling the nodes (or voxels) as either being a region of interest or being part of the background in the data. If there are various types of regions-of-interests in the data, then the labelled region of interest points can be further classified. In

scientific simulations, usually, a group of connected region of interest points form a meaningful (observable) scientific object. Examples of scientific objects are vortices in computational fluid dynamics, halos in cosmology simulations, or cell structures in blood flow simulations.

There are various techniques proposed to segment regions-of-interests including thresholding, region growing, clustering, machine learning algorithms, histogram based techniques, level-set techniques, model based, geometry and topology based techniques. A list of available techniques and their sample applications can be found in papers [1], [56], [36] and [86].

Recognition is the process of assigning a label to the detected region of interest from a set of available labels. In other words, it is identifying the type of an object. The recognition process is usually operates on a set of distinctive attributes (properties) of the region of interest. These attributes help distinguish feature types from one another. Typically, deriving such attributes requires the information of the object's boundaries (contours) so that by looking at the data values within the boundaries of an object various attributes such as the shape, volume and other attributes can be computed.

Domain knowledge is essential to build successful segmentation techniques. There is (currently) no known segmentation technique that work wells in each and every domain or applications. In many domains, a thresholding process yields effective results for feature extraction. In such applications, domain knowledge is utilized to determine the threshold value. However, in many scientific domains the thresholding operation by itself is not sufficient enough

and, therefore, is combined with the node (or voxel) connectivity information for a better accuracy in segmentation. The domain knowledge in such applications is used in both defining the threshold value and the type of connectivity. Region growing is one such technique. For example, in snake-based or level-set based techniques, the connectivity information expanded to include the “global” shape information of the object.

As the structure and the definition of the objects gets complex, the utilization of the detailed domain knowledge in the segmentation technique gets more important. In such complex techniques a new set of data, i.e., meta-data, is generated prior to the segmentation process. For example, in some techniques the mean and variance values are computed and saved in a vector form. Such *attribute* vectors are used in machine learning techniques to segment the region of interests or to identify the type of a region of interest. Besides the mean and variance values, there are many types of attributes defined and used in computer vision field for both action recognition and object segmentation. Recently, the power of computing local attributes in various applications has shown promise in various segmentation, detection and tracking applications. Examples of such local attributes are SIFT [45], meshSIFT [48] or SURF [6] techniques. For a detailed description of the SIFT technique see Appendix I.

In our proposed activity detection framework, any of the available segmentation and recognition techniques can be employed to extract features from scientific simulations.

2.2.2 Tracking

Since an activity is a temporal process that causes a change in an object's state, it is essential to detect the changes in each object's state and in each timestep. While each object's state can be computed from their spatial attributes in each timestep, as we start to process a new timestep, computers do not know which objects in new timestep respond to the ones from the previous timestep. (While correlating objects in time is a trivial task for the human brain and eye, this is a challenging task for the computers since the number of found objects, their location and shape information vary from one timestep to the next). The solution to this problem is widely known as tracking. In both computer vision and in scientific visualization fields there has been a considerable amount of work done for tracking regions-of-interests over time. Please refer to the surveys [91], [77] and [90] for a list of available tracking techniques and their applications in computer vision.

In scientific simulations, extracting and tracking scientific objects are considered together in time-varying data applications. These applications are usually called under the name *feature tracking*. The term *feature tracking* from scientific visualization and the term *object tracking* from computer vision applications have similar descriptions. They both fundamentally first segment the objects (features) in the data and then track (correlate) these segmented objects over time.

In scientific visualization, the first complete feature tracking framework was introduced by Samtaney et al. [1], as a solution to extract and track the

volumetric features within time-varying data sets. In their paper, they proposed the first feature tracking framework and used centroid position, volume, mass and 2D circulation information to track features. In the same group, Silver and Wang extended the feature tracking framework by introducing the volume tracking schema in [71], [72] and [73]. They observed that most features overlap between two consecutive time frames when the frame sampling rate is high enough. By incorporating this observation in their work, they also developed a more memory efficient algorithm which supports unstructured data sets. The next refinement, prompted by extending the feature tracking framework to a distributed Adaptive Mesh Refinement (AMR) data structures, allowed a viewer to isolate a multi-level isosurface and visualize its evolution spatiotemporally at different resolutions [1]. Next, the fitting of ellipsoids to the features was introduced for an efficient attribute computation and applied to the characterization and visualization of plumes [66]. Several studies use a predictive approach to tracking. In [64], Reinders et al. proposed a method that estimates the locations of the features in the next frame to improve the tracking quality by assuming that the features evolve predictably. Similarly, Muelder and Ma in [50] proposed a tracking schema including a prediction method. However, instead of extracting all the features first, they propose using the information of the current and previous frames to estimate the location of the feature in the next frame and they use that information to perform both segmentation and tracking at the same time. Other studies address different ways of abstracting the correspondence. Ji et al. [33] proposed a method to track features by using higher dimensional

isosurfacing. Thus, instead of extracting the isosurfaces from each time frame and then performing an overlapping test, they propose using the higher dimensional geometry to track user selected features in time by performing an isosurfacing process in 4D as opposed to doing it in 3D. In [79], Tzeng and Ma applied neural networks to find the region boundaries by estimating the transfer functions in the feature tracking framework. Sohn and Bajaj [74], proposed a method to compute the correspondence in time-varying contour trees. By applying the contour tree on feature tracking, they tracked user selected contours. In [38], Laney et al. propose to use Morse-Smale complex to segment bubbles in a hierarchical segmentation structure. Different attributes have also been used to correlate features. In [34], Ji and Shen proposed using earth mover's distance metric to track objects as an alternative to the volume overlapping or attribute based matching methods. In [1], Caban et al. proposed to use first and second order statistics with run-length matrices to capture textures and to distinguish them. Thus, by generating a feature vector, they perform a similarity search to find the best matches. In [24] Gezahegne et al. proposed a method that allows objects to retain their original labels for a couple of frames, thus the algorithm can detect bubbles going through each other instead of being classified as merged and then split. Weber et. al. proposed a technique using Reeb graphs to track features in [83].

Besides the individual objects (features), defining groups of objects and tracking them in many domains has also been the interest in many domains. Early studies of group tracking consider movement of features within radar data a

list of such papers can be found in [82]. Computer vision studies also address group tracking. McKenna et al. [49] proposed a computer vision based tracking system. The system utilized color to disambiguate occlusions and to qualitatively estimate the depth ordering and position during occlusion. The system tracks groups of people through mutual occlusions as they form groups and then separate. Gennari and Hager [23] introduced a general class of partitioning functions to define a group, and a set of rules to split and merge groups. Based on the group definition, they proposed a modified PDA estimator to track groups of objects. They reported that they can detect the groups of people that merge and split. Mucientes and Burgard integrated Multi-Hypothesis Tracking (MHT) with Murty's algorithm to track clusters of people [51]. Joo and Chellappa targeted on solving the data association problem in object tracking using the multi-hypothesis approach [35]. Lau et al. extended the MHT method to hypothesize over both, the group formation process (models) and the association of observations to tracks (assignments) [39].

While grouping and group tracking has long been studied in computer vision, it is relatively a new subject in scientific visualization. As part of this dissertation, a group tracking algorithm which is an extension of the feature tracking algorithm of Silver and Wang in [71] and [73] is described.

Any of these above-mentioned techniques can be employed in our proposed activity detection framework to track region of interests.

2.2.3 Activity detection

Activity detection is the process of searching for the existing instances of an activity in time-varying data sets. While there is a slight difference between activity detection and activity recognition, fundamentally, we will assume that they both serve the same purpose: extracting the instances of a certain activity or a set of activities. The subtle difference between activity detection and activity recognition lies in the definition of activity. If there are multiple activities are defined (modelled) in the data, then the process of finding and labelling the instances of any of these activities in the data is known as *activity recognition*. If there is only a single activity to detect (or if the purpose is finding “any” activity in the data regardless of its label), then the process of finding the instances of the activity is simply an *activity detection* process. Note that the terms “activity detection”, “activity recognition”, “action detection” and “action recognition” have also been used interchangeably, as discussed in the survey paper [78].

Activity detection is related to data mining. Data mining is the process of finding new and nontrivial information within a data set. This information can be in the form of a pattern, a model of the process that generated the data set or the correlation information between the available variables. Activity detection specifically deals with the complex patterns that are in the form of sequences (or combinations) of simpler patterns. A successful data mining technique requires domain knowledge [26]. This is also true for a generic activity detection framework. Inputs and outputs of a generic activity detection framework are shown in Figure 4.

While supervised and unsupervised techniques can also be used for activity detection, they do not provide an intuitive and easy way to express a hypothesis. Our focus in this paper is presenting a technique that allows a scientist to specify an event and search for it. Therefore, in this paper we propose to use Petri Nets to express and model an activity. A Petri Net model is a graph abstracting an activity. Therefore the same Petri Net model (graph) can be used in different simulations with different parameters. Compared to the learning based data mining techniques, their performance depends on the model description as opposed to forming new training data. Choosing between learning based algorithm and graph based algorithm balances a trade-off between needing additional data for training and needing to specify an accurate description. A graph based approach is more likely to meet our objective of enabling hypothesis testing.

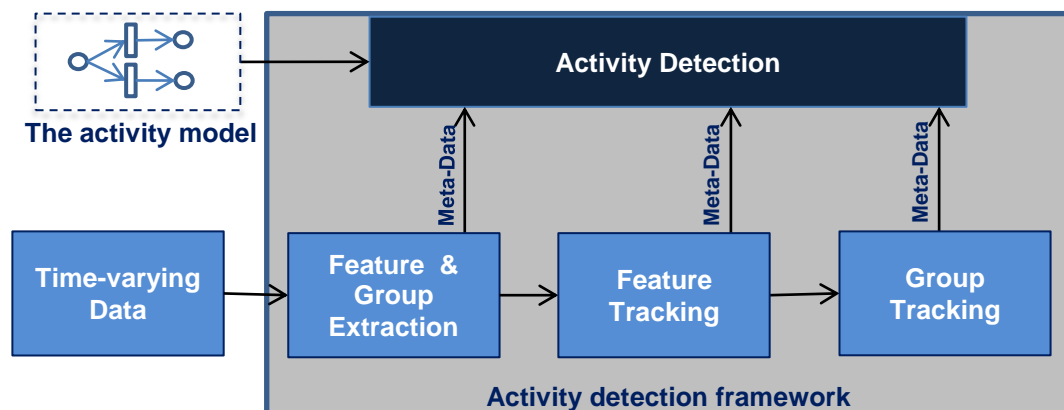


Figure 9: A generic activity detection framework. This figure emphasizes on the inputs of the activity detection module. Activity detection module runs on meta-data. The content of the meta-data depends on the domain and on the description of the activity.

As a framework, activity detection involves numerous steps including segmentation, tracking and computing the feature attributes. An illustration of

such individual steps and their connection to each other in a complete activity detection framework is given in Figure 9. In Figure 9, the entire data is processed only within the *Feature & Group Extraction* module. This step produces its own meta-data. Tracking modules use this meta-data as input and then create their own meta-data as output. Activity detection module operates on only the meta-data that is computed in both extraction (segmentation) and in tracking steps.

In the next chapter, we review the related activity detection work.

CHAPTER 3:

BACKGROUND – RELATED WORK

This chapter presents the related work from the computer vision field.

Activity detection (or activity recognition) has been studied in many fields under the names event detection, event recognition, activity detection or activity recognition. Figure 10 classifies a list of various activity detection techniques. However, since computer vision applications are the most relevant applications that can be applied to scientific visualization, the focus in this chapter is on computer vision applications.

3.1. Related work in computer vision

In computer vision, the activity detection problem is treated with different approaches including clustering, machine learning or semantic based techniques using rules or graphs such as Petri Nets. All of these approaches require the domain knowledge in different forms. For example, in the case of machine learning, the domain knowledge is embedded within the training data. The learning process and the accuracy of the technique depend on the validation. Validation requires ground truth which is a set of expected or real outputs (labels) from the technique. Picking (or generating) the ground truth or the training data is equivalent to manually picking and labelling multiple instances of the activity within the data set. This is almost the same as exploring the data manually in each timestep and, therefore, is an extremely time consuming task [62].

Semantic based activity detection approaches simply generate a model of the given activity from its description. These approaches do not require training data or a certain type of cost or similarity function to be derived. Instead, the domain expert defines the activity in a sequential form in a timely manner and this description is used to search for the event. The sequential form can be in the form of a set of rules (as in [75]) or a graph (as in [25], [40], [58], [42]). Both rule-based and graph-based techniques are fundamentally logic based (if-then based) techniques. Among those, the graph-based techniques use a state based approach in which the various stages of an activity are described as the individual nodes (e.g., finite state machines, Petri Nets). Petri Nets encompasses both rule-based techniques and finite state machines (a finite state machine is a subclass of Petri Nets. See Chapter 5). Using the Petri Net formalism, a scientist models an activity in a graphical fashion where objects in the simulation pass through different stages on the way to being classified as an activity. Once the graphical model is available, a Petri Net algorithm evaluates the model to search for the instances of the activity automatically. (This is analogous to knowledge-assisted visualization [16].)

In computer vision, activity detection applications mainly focused on human related activities. These include interactions or relations between: humans and humans [54], humans and vehicles [31], humans and web sites [85] or certain human behaviours or their situations in certain environments as in [76]. Various techniques including Bayesian techniques, hidden Markov models and conditional random fields are used to “learn” and detect activities in such

examples. A detailed list of available activity examples and detection techniques can be found in the review papers [1], [61], [40] and [78]. Recently, Petri Nets gained the attention of researchers in both data mining and activity detection communities as in [3], [14], [25], [40], [58] and [42]. This is due to the fact that Petri Nets can be used as a natural way of modelling semantic descriptions of activities or events.

While the activity detection related Petri Nets works aimed to work with video data, they do not incorporate the “dynamic” properties of the time-varying environment (e.g., split, merge, appear and disappear events) within the Petri Net formalism. Some of the above-mentioned Petri Net treatments such as timed Petri Nets or Stochastic Petri Nets still lack supporting the dynamics of a “time-varying” system in the sense that the objects (with their attributes) change in time-varying systems.

Figure 10, classifies various techniques based on their distinctive properties. Firstly, all the techniques are categorized according to their purpose: whether the technique is used for a search related purpose or for a knowledge discovery (data mining) purpose. Knowledge discovery techniques fundamentally rely on the anomaly detection principle, in which, typically, data is “clustered” first based on a predefined metric and then the objects that fall in different clusters are further investigated usually by analysing the relative frequency of the occurrences in each cluster. Example applications can be found in [18] or in [62]. Unless the domain knowledge is properly utilized, clustering based techniques

(with the use of commonly available metrics) do not guarantee finding what the user/scientist is interested.

Search based techniques all include the domain knowledge in a certain form. The domain knowledge is the information of the activity being searched and of the environment in which the activity is performed. Search based techniques can be further classified based on how the domain knowledge is utilized. The activity being searched can be given as a part of the data (with labels attached to individual time steps). This is the most common form of presenting the template of the activity in activity detection applications. Example approaches of modelling the activity in data form can be found in [63] or in [89]. Alternatively, the definition of activity can be given semantically as in [3]. Semantic approaches typically express the entire activity as a graph in which the order of the nodes are drawn from the given semantic description. On the other hand, machine learning based techniques learn the order of states and their numbers typically from the given data. In some domains, the activity can be modelled with a mathematical function as well.

While the goal of a typical activity detection framework is fundamentally extracting all the instances of the given activity, some of the available techniques (usually the early activity recognition works in computer vision) work on detecting a single instance (the first available instance or a random subset of all the available instances) of the activity in the given dataset.

Modelling an activity is important. In the literature, while some work focused on modelling activities of a single agent (an activity performed by a single object,

such as a single human), other work focused on modelling activities performed by multi-agents (the activities of multiple agents, such as multiple humans) in the literature. Notice that multi-agent activity detection techniques do not necessarily work on detecting all the instances of the modelled activity in the data. For example, the model given in [58] focuses on detecting a single activity performed by multiple people.

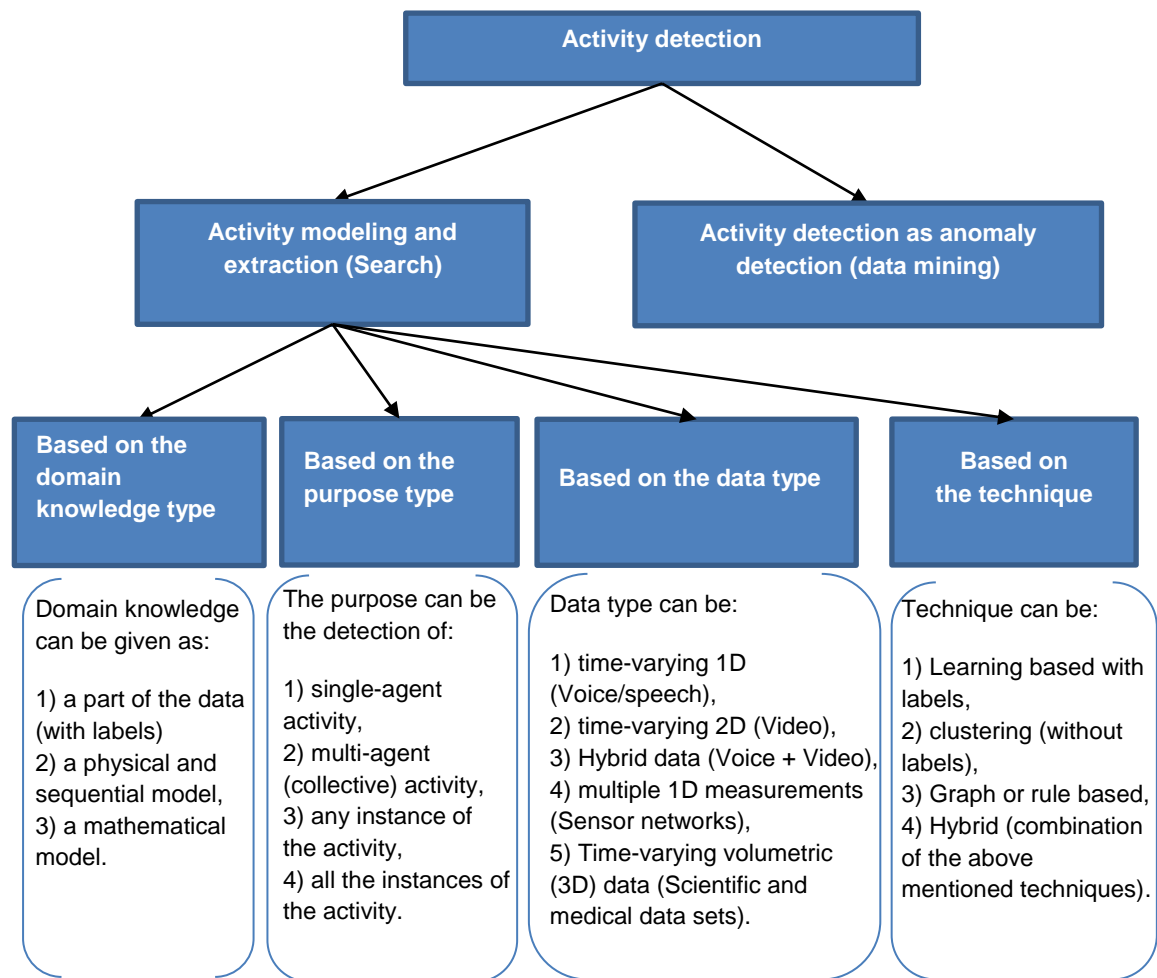


Figure 10: A classification schema for various activity detection techniques. Rather than listing specific techniques only, this classification schema also aims to give a sense of how the available techniques can be classified based on their input or outputs.

The modelling strategies also vary based on the spatial properties of the domain or the dimensions of the data. Some modelling techniques utilize the spatial or topological structure or the spatial/topological change that is inherent in the data.

3.2. Related work in visualization

In scientific simulations, there are many cases where the scientists are interested in analysing complex events or features. Examples are: formation of a packet [65], formation of a galaxy as in [9] and combustion events as in [84]. Three specific examples we use in this paper include “Anomalous Plume Bending”, “Packet formation” and “Merge-Split”. Many other domains have the need for detection of activities such as blood clot formation in blood flow simulations [30], magnetic storm formations in space-weather simulations [46] and extinction and re-ignition in turbulent flames in combustion simulations [47]. All these examples involve actors performing activities. Therefore all these activities can be modelled by using a common formalism. Activities occur over multiple timesteps in which feature attributes or their types may change over a course of an experiment / simulation.

Activity detection techniques work on a set of computed attributes. This assumes that both the regions of interests are already segmented, tracked and their attributes are available. Therefore, in this section first we will briefly go over the available activity detection work in video visualization and then will go over the techniques that could help segment and track the objects in time-varying 3D scientific data sets.

Video Visualization: Event visualization in video data has been widely studied in the visualization community. For example, Botchen et al. presented a video visualization technique, VideoPerpetuoGram, for action visualization in video data [11]. In their method they treated the stacks of 2D time-varying video data as 3D volume data and visualized actions in such volume data. Parry et al. presented a hierarchical event selection algorithm for event visualization in video data and applied their method on snooker video [55]. A list of available applications and techniques can be found in the survey paper [10]. However none of these papers had a technique to model a scientific activity.

Related work in scientific visualization includes extraction and tracking. Extracting atomic events and features from time-varying data has been widely studied in scientific visualization. This information is crucial to activity detection and provides the meta-data input to an activity detection framework.

3.3. Activities as patterns and the use of machine learning

Learning techniques requires labels attached to the data. In (machine) learning techniques, labels are formed first. This is usually done by isolating the time interval involving a sample activity in the data first. Next step is isolating the objects that are part of the activity within the isolated time interval of the data. Then, each object's certain and *distinctive* attributes are computed in each timestep within the isolated data. This process is repeated for multiple instances of the same activity, and also for other activities to create a library of activities. Each activity sample is assigned a label. This process can also be considered *creating the training data* for machine learning techniques.

The success of the pattern recognition techniques is highly correlated to whether the selected attributes sufficient enough to discriminate the activity to be detected from all the other activity types in the data. To ensure that, some techniques first compute N possible attributes and then selecting only the M of these N attributes where $M < N$ (or preferably $M \ll N$) by incorporating the domain knowledge. The other techniques compute only the “smart” (i.e. discriminative) features by using the domain knowledge.

It is an active research field to compute and define the useful and discriminative attributes for activity detection. These attributes can be only spatial or spatio-temporal. Spatial attributes concern only the object’s current state and therefore they can summarize and characterize only the current state of an object. Spatio-temporal attributes summarize and characterize a series of an object’s state over a certain time interval. Current trend in computer vision is studying individual activities and summarizing these activities manually.

CHAPTER 4:

GROUP TRACKING

Group tracking is an integral part of activity detection. In this chapter, we extend feature tracking to extract, track and follow groups of features as they travel and interact in a time-varying data set.*

In many scientific domains, physical groups exist besides the individual features. Understanding both the individual feature evolutions and the group evolutions is important to understanding the dynamics of scientific phenomena. For example, groups of cells and studying their collective behavior is of interest in biochemistry [57], groups of galaxies (halos) is of interest in cosmology [9] and groups of hairpin vortices (packets) is of interest in wall bounded turbulent simulations [53].

Generally speaking, a group is a set of coherent structures (features) that are related to each other in certain ways or “act” together. (This is analogous to birds that both act individually and fly together in flocks). An example from scientific simulations is a pair of features where a feature rotates around the other feature. Therefore group extraction and group tracking is as important as feature tracking in many scientific simulations. All the above mentioned group examples have a hierarchical structure where a set of smaller structures (e.g., a set of voxels) forms a high level structure (e.g., a feature) and a set of these high level

*Some material presented in this chapter has been published in an LDAV 2012 paper:
S. Ozer, J. Wei, D. Silver, K.-L. Ma, P. Martin, "Group Dynamics in Scientific Visualization", Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on, 2012.

structures (e.g., features) forms an even higher level structure (e.g., groups). This is illustrated in Figure 11.

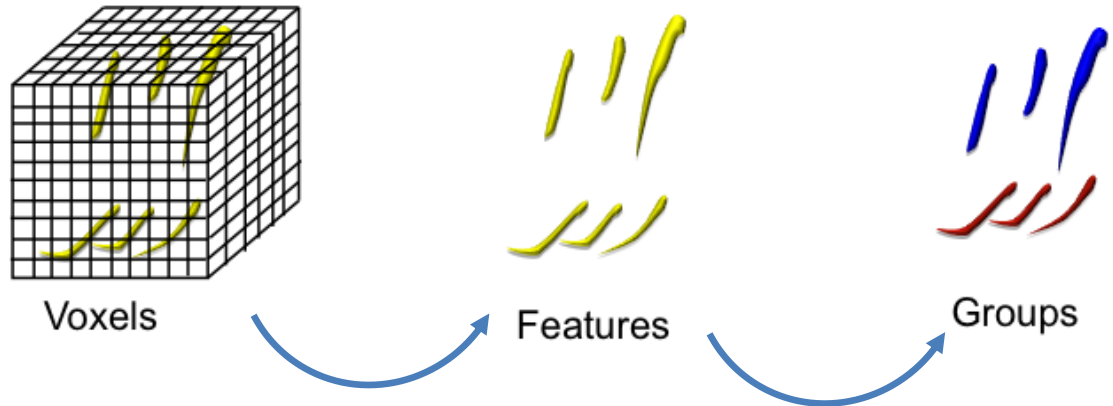


Figure 11: An example of a hierarchical structure. A voxel is an atomic structure for a feature, and where a feature is the atomic structure for a group.

While feature tracking follows the interaction and evolution of individual features, the dynamics of groups of features has not been fully addressed in scientific visualization. Therefore one of our contributions in this dissertation is presenting a group tracking schema for time-varying 3D scientific simulations. Our focus is on unconnected or disambiguated features in this dissertation (although, our model can also be applied to groups of connected features). While many features in scientific simulations can be connected at very low thresholds, our goal in this work is to understand the dynamics of groups of individual features.

In order to track groups, we first extract features, track them and then group them based on the domain specific grouping criteria. Once features are grouped at each timestep, the groups are tracked to see how they change and interact over time. A clustering algorithm is employed to cluster (group) the features. One

benefit of employing a clustering algorithm is the ability to isolate and map the group definitions from various domains to the computational domain via similarity functions. This property of clustering algorithms helps us to build a generic model for group tracking. We apply our group tracking model on a wall bounded turbulence-Direct Numerical Simulation (DNS) and demonstrate various visualizations in the following subsections.

4.1. Overview of the group tracking algorithm

The group tracking framework flow diagram is shown in Figure 12. In this diagram, the first step is feature extraction (Figure 12a). Feature extraction is discussed in Chapter 2 and Chapter 3. Once features are extracted, their spatial attributes can be computed. A spatial attribute is an attribute that can be computed by using only a feature's segmentation information. Table 1 lists various examples of spatial attributes.

Spatial feature attributes	Time-dependent feature attributes	Feature-to-feature attributes
Centroid, Max/min, Volume, Shape, Total number of particles, Mass, Self-Orientation, Moments, Extends, Mean/variance over voxels, Surface information.	Velocity, Acceleration, Self-rotation, Swirl, Extension, Expansion, Shrinkage, Change in a feature attribute, Mean/variance over time.	Rotation around a feature, Distance, Relative orientation, Nearest-neighbour information, Variable difference/sum, Mean/variance over features.

Table 1: Three different attribute categories and examples for each category

Next step is feature tracking (Figure 12b). Feature tracking correlates the features from the previous timestep t_{i-1} to the next step t_i [72] (See Chapter 2 and Chapter 3 for a list of available feature tracking techniques). This correlation

information is saved in a feature history. In this step, time-dependent feature attributes can also be computed (time-dependent attribute examples are given in Table 1). These are the attributes derived by jointly considering the current and previous spatial-feature attributes. They can also define the change in a feature attribute over a specified duration. Accessing a specific feature's attribute in the previous timesteps requires tracking information, i.e., feature history. An example of this is the velocity. Other attributes are listed in the middle column of Table 1.

Concurrently with feature tracking, one can also compute feature-to-feature attributes. These are the attributes that are computed by comparing a feature to neighboring features in the same timestep. They can be derived from feature attributes or time dependent feature attributes. For example, the mean (or variance) of a feature attribute can be computed over the neighbor features.

In the next step groups are determined (Figure 12c). A group is a set of coherent features that are associated together based on some criteria. These criteria can be expressed in terms of all the feature attribute types. A list of sample attributes can be found in Table 1.

Grouping criteria can be geometry, distance, shape, rotation or orientation based. For a group, all these criteria can be combined and expressed within a single “similarity” function which is used by a clustering algorithm to determine the groups. Once groups are determined, spatial group attributes can also be computed in this step. Spatial group attributes are defined similar to the spatial

feature attributes. They summarize the spatial properties of the extracted group and its member features.

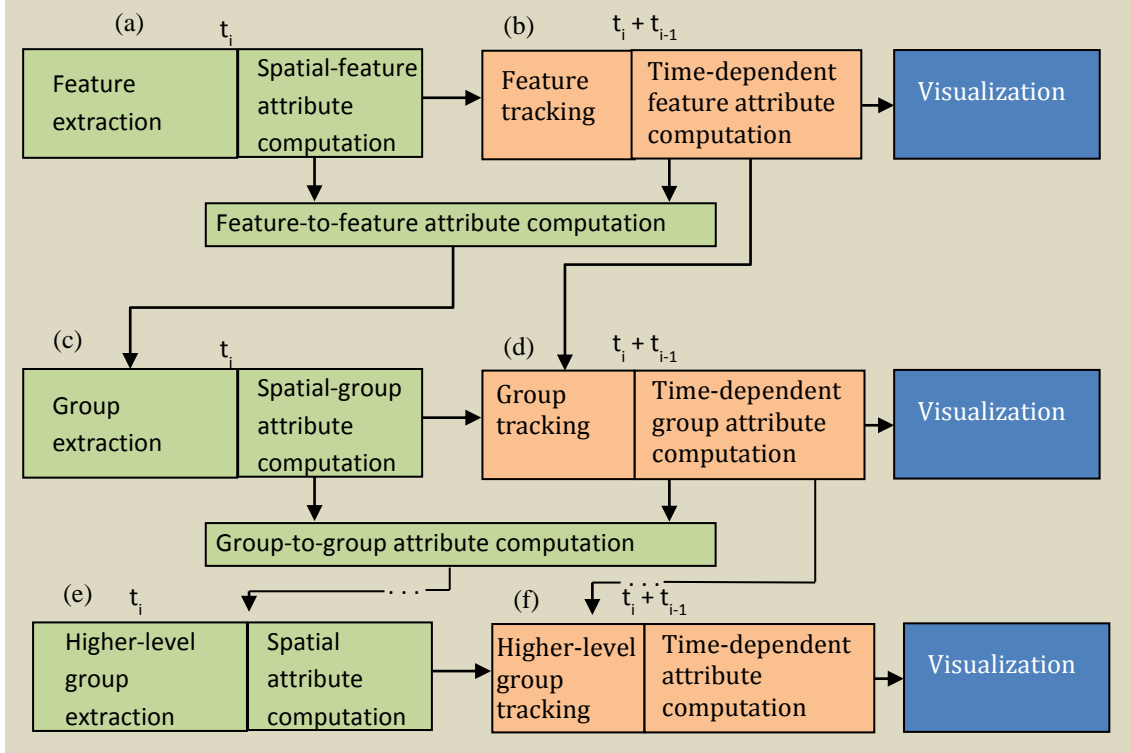


Figure 12: Group tracking framework flow diagram.

Tracking groups is the next step (Figure 12d). Group tracking correlates the extracted groups in timestep t_i to the groups in timestep t_{i-1} . Once computed, this computed correlation information is saved as group history. Time-dependent group attributes are also computed in this step by using the group history. Time-dependent group attributes are defined similar to the time-dependent feature attributes. These are the attributes derived by using the current and previous spatial-group attributes.

Once groups are determined and tracked, Group-to-group attributes can be computed. These attributes are also defined similar to their feature counterpart,

i.e., feature-to-feature attributes. They are the attributes that are computed by comparing a group to one another or to a certain number of neighbour groups in the data. The last steps involve creating super-structures of groups (Figure 12e). If the domain has super-structures (groups of groups) that are defined with a different set of criteria, then similar to the group extraction step, super-structures can be extracted by using the related similarity function for the super-structures. At this step, each group is assigned to a super-structure by a clustering algorithm. Spatial super-structure attributes are also computed at this step. These attributes are also defined similar to the spatial-feature attributes. They summarize the properties of the super-structure and its member groups. Furthermore, time-dependent super-structure attributes can also be defined similar to the time-dependent feature attributes. These are the attributes derived by using the current and previous spatial- super-structure attributes.

Recursively, once defined, all the higher level structures can be extracted and tracked in a similar way to the super-structures. And their associated attributes can be computed.

Time-varying visualization uses the tracking results at each structure level. Therefore once the tracking is performed and the related time-varying attributes are computed, the associated level structures can be visualized. In the following sections, we will focus on clustering, group tracking and group visualization in detail.

4.2. Group extraction via clustering

Extracting groups, fundamentally, is a process in which each feature is assigned a group ID. Each feature in the same group would have the same group ID while each group would have its own unique group ID. Grouping can be considered as a clustering problem. Typically, a clustering algorithm runs in an attribute space in which a point would represent a scientific feature. In such an attribute space, a feature is represented with a vector formed of feature attributes. A list of different types of feature attributes is given in Table 1. Therefore it is essential to compute the feature attributes that would help describe a group before starting the clustering process. Clustering algorithms usually operate with the assumption that closer points in a feature space fall in the same cluster. Therefore it is essential to convert a scientific group definition in a metric such that the metric should give a closer value for the same features in a same group. If this metric is a similarity metric, then as the similarity value gets greater, the features become more similar in the attribute space. If Euclidian distance is used for the metric, then the smaller the distance value, the closer the features and as the features get closer, they start to fall into the same cluster.

Once features are extracted and their feature-to-feature attributes are computed, they can be grouped using a clustering algorithm. The focus below is on simulations where features cluster together and then act in groups. Hierarchical and partitioning clustering algorithms are two main types of similarity based clustering methods [32], [88]. Hierarchical clustering seeks to build a hierarchy of clusters. It either starts from each individual data object as a cluster

and merges two most similar clusters every time until only one cluster is left (agglomerative clustering); or it starts from the whole collection of data as a cluster and split the data set recursively until reaching a pre-specified cluster number. Partitioning clustering, such as the K-means algorithm, divides data objects into a number (often specified by a priori K value) of clusters according to some optimization criterion. Hierarchical clustering yields a hierarchical structure besides the final cluster information. Hierarchical clustering does not require a pre-specified number of clusters as opposed to the partitioning clustering. Moreover, hierarchical clustering does not require any initialization parameters as opposed to K-means algorithm that require initial cluster centroids. However, there is a trade-off between these advantages of hierarchical clustering and its computational efficiency. For more information on the clustering algorithms please refer to the text books [4] and [27]. In this work, we use hierarchical clustering to extract groups in our group tracking model. At each timestep the clustering algorithm is run and groups are formed.

In a clustering algorithm, the features with the highest similarity are grouped into the same cluster. Notice that within a cluster, all the features should be similar to each other, while each of them should be dissimilar for the inter-cluster features. The similarity function that is used in the clustering algorithm is defined and described in the next sub-section.

4.3. Similarity functions

A similarity function is a function $S: R^n \times R^n \rightarrow R$ that provides a measure of the similarity between two given vectors in a similarity space. These two given (input) vectors represent two different features with their n attributes. For any two given vectors \mathbf{F}_A and \mathbf{F}_B , the similarity measure $S(\mathbf{F}_A, \mathbf{F}_B)$ is the same as $S(\mathbf{F}_B, \mathbf{F}_A)$, i.e., $S(\mathbf{F}_B, \mathbf{F}_A) = S(\mathbf{F}_A, \mathbf{F}_B)$. Moreover, $S(\mathbf{F}_B, \mathbf{F}_A) \leq S(\mathbf{F}_A, \mathbf{F}_A)$ and $S(\mathbf{F}_B, \mathbf{F}_A) \geq 0$. Notice that if the vectors \mathbf{F}_A and \mathbf{F}_B are dissimilar, then the similarity measure should be low. While in some work, the similarity value can have negative values; it is assumed to be nonnegative in this dissertation.

In our applications, we assume that if the vectors are dissimilar, then the similarity value between the two vectors is the minimum value (i.e., 0 in our applications), i.e., $S(\mathbf{F}_B, \mathbf{F}_A) = 0$. With these conditions in mind, a group can be defined in terms of a similarity function. This similarity function will have a nonzero similarity value for any two given features in a given group and will have a zero similarity value for any given two features from different groups. For example, distance based similarity functions yield a higher similarity value as the distance between two input vectors decreases; i.e., as the features get closer to each other, the similarity value increases. Another example can be a shape and distance based similarity function in which the features that are close to each other and look alike would give higher similarity values. A threshold can be set for the similarity function value to define the terms similar and dissimilar. A higher value of the similarity function implies a greater similarity between the given two features. Once the similarity function is defined or selected from an available

list/library, the clustering algorithm can group the features based on this provided similarity function.

4.4. Group tracking & group events

Identifying the dynamics of structures (e.g., features or groups) requires correlating these structures over time, which is generally known as the correspondence problem [72]. The tracking step addresses the correspondence problem in the group tracking framework (in Figure 12d). Once we have identified groups and features in the data, we can identify what happens to groups over time. Similar to features, groups can merge, split, appear (birth), disappear (death) or continue (see next section for details).

In the framework, group level tracking (matching) from one timestep to the next can be performed by combining the feature tracking information of the current time with the extracted group information (as shown in Figure 12d). Clearly, if the features overlap in volume, then their groups also overlap in volume. Therefore, in this work we employ a volume overlapping schema to track groups.

Even in the cases where the features do not overlap, their groups can overlap since groups are bigger structures. In our model, each group has a list of its individual features and each feature has its *group_ID*. For reference, each feature is represented by $F_{k,j}^i$, where k is a unique feature identifier, j is the group identifier that the feature belongs to and i is the timestep (t_i) index. Similarly, each

group is represented by G_j^i , where j is a unique identifier in the timestep t_i . A particular group is the union of its member features:

$$G_j^i = \bigcup_{k=1}^h F_{k,j}^i \quad (1)$$

where h is the number of total features within the group with the identifier j in timestep t_i . We use the following definition for volume overlap for groups which is an extension of the one in [72]:

Overlap: If the group G_A^i corresponds (matches) to G_B^{i-1} , then G_A^i overlaps G_B^{i-1} . i.e., $G_B^{i-1} \cap G_A^i \neq \emptyset$.

By using this overlap definition, an overlap table can be computed between the groups from one timestep to the next. The overlap table can be computed by using various criteria. These can include:

Feature overlap criterion: This criterion uses the sum of overlapping feature volumes between the groups. Since we assume that a complete feature is a part of a group, a group's volume can be computed by summing up the volumes of each member features. Figure 13 illustrates the tracking process with the feature overlapping criterion. The feature and group extraction steps (with attributes) are completed in the first timestep t_1 . When the framework starts processing the next timestep t_2 , the first step is extraction of the features and computing the feature attributes. Then features are matched to the ones in t_1 by using the volume overlapping criterion [72]. Feature tracking relates features from t_1 to the ones in t_2 . Assume that grouping is done based on the distance information only. In this case, groups can be extracted by using the spatial-feature attributes only. Group extraction yields GroupA, GroupB and GroupC in t_2 . Group tracking step creates

the overlap table based on the feature overlapping criteria (instead of using the actual values computed in overlap table for feature tracking). For groups, overlap table is computed by using the entire volume of the joint member features between t_1 and t_2 . This is shown in Table 2.

	Group_1	Group_2
GroupA	F_A	0
GroupB	F_B	$F_C \cup F_D$
GroupC	0	F_E

Table 2: Overlap table by using the feature overlap criterion.

Based on the overlap table, GroupA is matched to Group1 and GroupC is matched Group2 only. However GroupB is matched to both Group1 and Group2. The dominant group for Group B is Group1 since the volume of F_B is greater than the sum of volumes of F_C and F_D .

Feature number overlap criterion: This criterion uses the total number of overlapping features within each group. If the features are relatively homogeneous in volume and shape, then this kind of a simplification could work reasonably faster. However, matching/correspondence is a function of both time and spatial properties. Using only the number of matching features does not summarize the spatial attributes of groups properly since it ignores the volume or shape information and thus may not yield an accurate matching. Especially in cases where a threshold is used, the least matching groups with the least feature numbers can be ignored. However such groups might have bigger volume and might be the actual dominant matching groups.

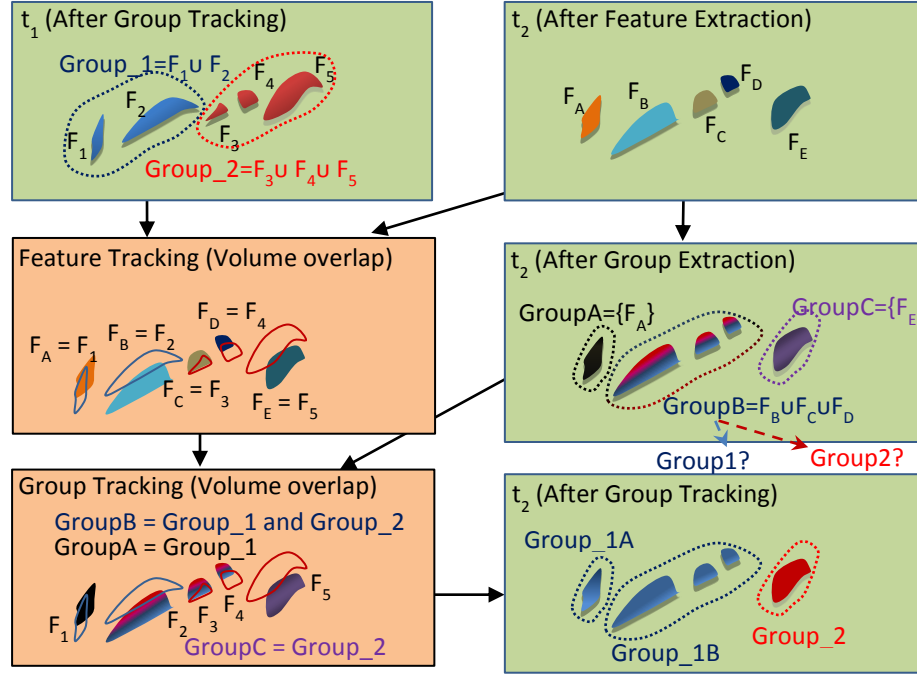


Figure 13: An illustration of volume-overlap based group tracking. GroupB is formed of two features from Group₂ and one feature from Group₁ in t_1 . This is a partial split event. Groups in t_1 and t_2 can be correlated visually after the group tracking process.

Convex hull overlap criterion: This criterion first defines a convex hull for each group that comprise all the features within, and then perform a volume overlap test between the convex hulls for group matching. This criterion considers the spaces between the features as well. The volume of the containing envelope of all the features in a group is used. Thus, a convex hull of an oddly shaped group formed of 2 or 3 small features can contain a bigger volume and can overlap with other neighbour groups.

In our case study, the features are not homogenous in volume or in shape. Therefore, the feature overlap criterion is used in our applications. A tolerance value can be used in an overlap table to ignore smaller overlaps [72]. This

process helps eliminating unwanted small amounts of overlaps (matches) or detecting false events. Groups G_A^i and G_B^{i-1} are matched, if:

$$\frac{\max(G_A^i - G_B^{i-1}, G_A^i - G_B^{i-1} - G_A^i)}{\max(G_B^{i-1}, G_A^i)} < Tolerance \quad (2)$$

Equation 2 defines the normalized volume difference test as in [72]. The domain dependent *Tolerance* value is a percentage value. Figure 18 shows an example visualization of both feature tracking and group tracking.

4.5. Group & cross-level events

Feature tracking can characterize the evolutionary events (dynamics) of features such as merge, split, continue, birth (appear) and death (disappear) [72]. Similar to features, the higher level structures (groups) can split, merge, continue or die. However, these group events are slightly different than the ones defined for features. The difference is due to the fact that the features are unconnected in groups. A list of defined group level events is illustrated in Figure 14.

Birth: A new group of features is formed in the current timestep and is not correlated to any group in the previous timestep. i.e., if the group G_A^i does not overlap any groups in t_{i-1} ($G_B^{i-1} \cap G_A^i = \emptyset$), then the group G_A^i is considered as a new born group.

Death: An existing group of features in the previous timestep disappears in the current timestep. If the group G_B^{i-1} does not overlap any groups in t_i , then the group G_B^{i-1} is considered as a disappearing group and the event is called a death event.

Full Split: A single group G_B^{i-1} in t_{i-1} splits into a number of N groups ($N>1$) in t_i . Each of these group in t_i overlaps G_B^{i-1} , i.e., $G_B^{i-1} \cap G_j^i \neq \emptyset$ for each $j \in N$.

Full Merge: This is the event where a number of N groups merge to form a single group. i.e., if a number of N groups ($N>1$) in t_{i-1} merge to form a single group G_B^i in t_i , then G_B^i overlaps each of N groups in t_{i-1} , i.e., $G_B^i \cap G_j^{i-1} \neq \emptyset$ for each $j \in N$. An illustration of full merge is shown in Figure 9b.

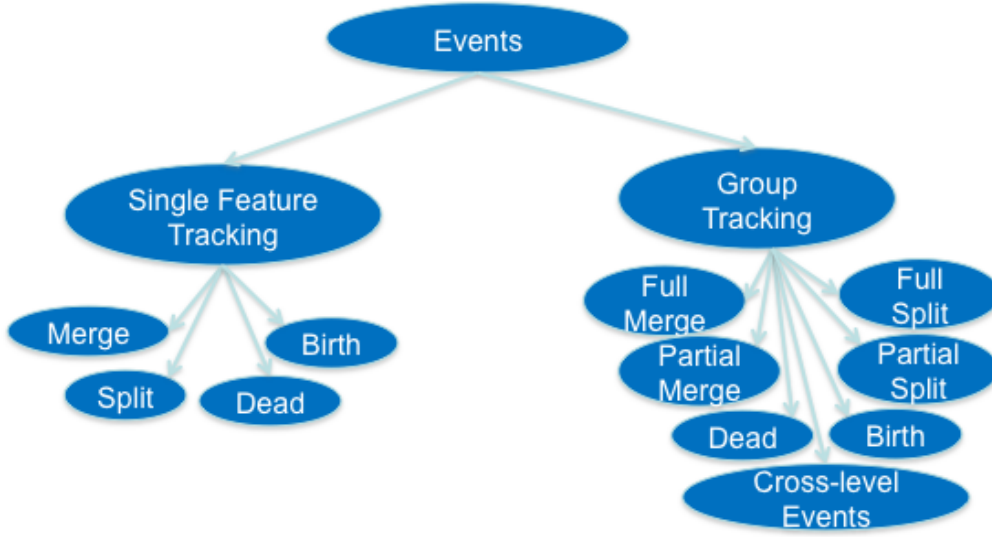


Figure 14: Single Feature Track events vs. group tracking events. In group tracking, merge and split events are further divided into being full or partial events. Moreover, cross-level events can be defined within a group tracking schema.

We define the following events for groups:

Partial Merge: This is the event where some features from several groups (portions of several groups) leave their main groups and join other existing groups resulting in a lesser number of total groups. Different portions of a group in t_{i-1} can merge to several other groups in t_i . Also note that different portions of different groups in t_{i-1} can merge to the same group in t_i . Therefore the total number of N groups in t_{i-1} merge and form a number of M new groups ($M<N$) in t_i .

where each of these N groups in t_{i-1} overlaps each of the M groups in t_i . At the end, the total number of groups M in t_i will be less than the total number of groups N in t_{i-1} . An illustration of partial merge is shown in Figure 9a.

Partial Split: This is the event where the portions of a number of N groups split from their groups to “get together” and form a new group. i.e., only the portions of a total number of N groups ($N > 1$) in t_{i-1} get together and form a total number of M new groups in t_i . In this case, each of these N groups in t_{i-1} , overlaps each of the M groups in t_i . At the end the total number of groups M in t_i will be higher than the total number of groups N in t_{i-1} .

Continuation: If none of the above-mentioned events occurs, then this is a continuation event for the group.

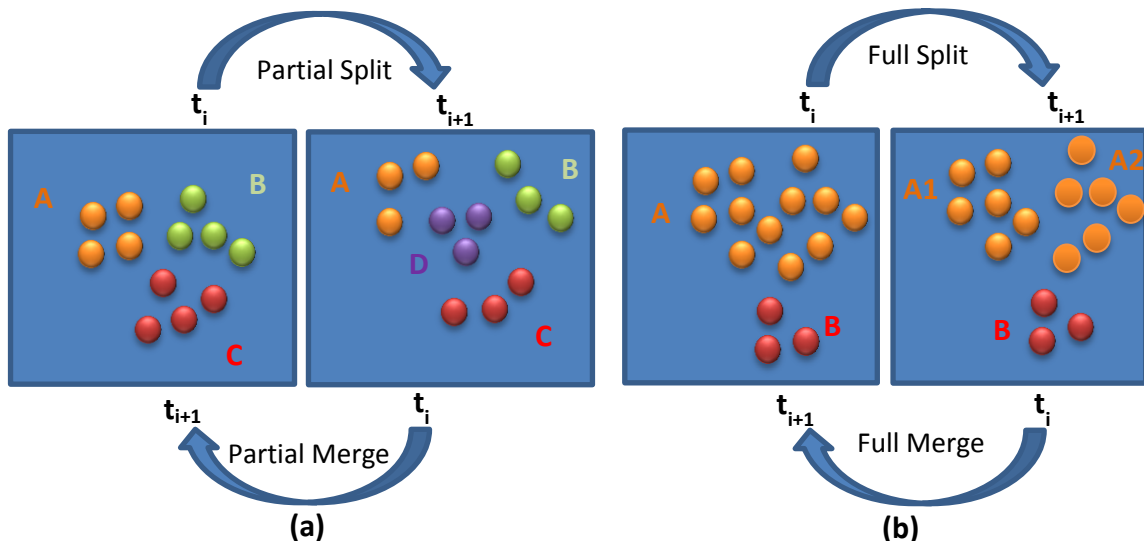


Figure 15: An illustration of full merge, partial merge, full split and partial split. (a) partial merge and partial split, (b) full merge and full split events.

Figure 15 illustrates these events. As Figure 15a illustrates, if multiple groups (say N groups) in timestep t_i partially correlate with multiple groups (Say M groups) in the timestep t_{i+1} and if the number of correlating groups N differs than

the number of the correlating groups (M) in the next timestep, then we say that there is a partial event happening between those two timesteps. Depending on the case whether $M > N$ or $N > M$, we call the event *Partial Merge* or *Partial Split* respectively. If M is equal to N , then we will consider that there is no split or merge event happened at the group level, therefore this case (where $M = N$) is called *continuation* event.

Figure 15b illustrates the *full split* and *full merge* events. If a single group in the timestep t_i correlates with M groups in the timestep t_{i+1} , or if N groups in the timestep t_i correlates with a single group in the timestep t_{i+1} , then we say there is a full event occurred between the timesteps t_i and t_{i+1} . If the event is from 1 group to M groups then we call the event as *full split*, and similarly if the event is happening between the N groups in t_i and a single group in t_{i+1} . Then we call the event as *full merge*.

As it can be inferred from the above definitions, *partial events* are the ones happening between many groups and many groups; *full events* are the ones happening between a single group and many groups. In addition to these events, when hierarchical structures are considered, the cross-level events can also be defined and detected. Cross-level events are the events that happen between a structure and a higher level structure. For example, group tracking allows detecting a feature leaving its group to join another one (cross-group event).

Cross-group: This is the event where a feature leaves its group to join another group. In this event, while at the group level, groups can remain the same (groups continue); at the feature level, a feature can move from one group to

another and therefore, this event is different than the partial merge or split events. An illustration of this event is shown in Figure 16. In the illustration, when we use the first level (feature) tracking, we only detect the split event that occurs within the Group_S. However, when the group tracking is used, we can detect that a feature moves from Group_R to Group_S (a cross-group event).

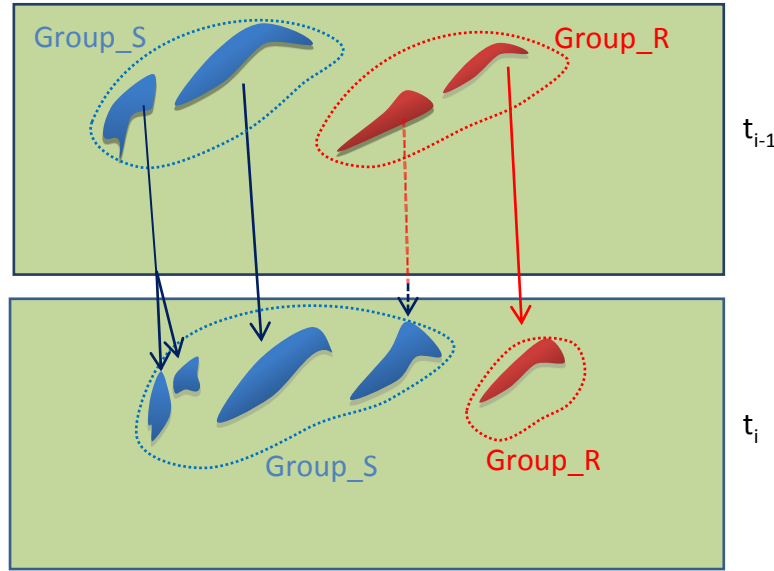


Figure 16: An illustration of cross-group event. A feature in timestep t_{i-1} moves from Group_R to Group_S in the next timestep t_i , while the number of groups remains the same.

4.6. Creating a domain specific similarity function

A packet is defined geometrically and this definition is based on the distance and the angles (orientation) between the features. Here, we demonstrate how to derive a domain specific similarity function which defines a scientific group. Our specific scientific group example is *packet* which is a specific term used in wall bounded turbulent simulations and it defines a group of hairpin vortices that move coherently.

Specifically, we define a packet as a set of features meeting all of the following three criteria:

- I. The distance between a given set of two features should be less than a predefined distance (threshold).
- II. The angle between two given features should be equal or less than 45° .
- III. The packet should be elongated along the X axis such that the cross section (in y-z plane) increases along the X axis.

Various distance measures can be computed and used for the first criterion, such as the nearest neighbour distance (on the feature surface), the centroid distance or the distance between the local extrema points. We notice that the local maximum point of each feature localizes around the top of the feature. Therefore we use the distance between the local maximum points to simplify the computations. We used different thresholds parallel to each axis for the distance criterion.

The function should provide a positive value for a pair of features in the same packet and a zero value for a pair of features from different packets. In this example to simplify the computations, we use only the coordinates of the maximum node value of each feature and construct 3 dimensional attribute vectors for each feature. With the assumption where $\mathbf{a} \neq \mathbf{b}$, for given any two features \mathbf{a} and \mathbf{b} , we define the distances along x, y and z coordinates as: $\Delta_x = a_1 - b_1$; $\Delta_y = a_2 - b_2$; $\Delta_z = a_3 - b_3$, where a_i and b_i ($i=1,2,3$) are the i^{th} element of the attribute vector.

Input: $a_1, b_1, a_2, b_2, a_3, b_3, \text{BoxLength}_x, \text{BoxLength}_y, \text{BoxLength}_z$ **Output:** $S(\mathbf{b}, \mathbf{a})$

Define the variables:

$$\Delta_x = a_1 - b_1; \quad \Delta_y = a_2 - b_2; \quad \Delta_z = a_3 - b_3,$$

$$D_x = \text{BoxLength}_x - \text{abs}(\Delta_x); \quad D_y = \text{BoxLength}_y - \text{abs}(\Delta_y); \quad D_z = \text{BoxLength}_z - \text{abs}(\Delta_z);$$

Define A, B and C variables as follows:

$$A = (\text{sign}(D_x) + 1) (\text{sign}(D_y) + 1) (\text{sign}(D_z) + 1) / 8; \quad B = (\text{sign}(\Delta_x \Delta_z) + 1) / 2;$$

$$C = (\text{sign}((\pi/4) - \text{atan}(\Delta_z / \Delta_x)) + 1) / 2;$$

The similarity measure is computed by the following equation (assuming that $\Delta_x \neq 0$):

$$S(\mathbf{b}, \mathbf{a}) = S(\mathbf{a}, \mathbf{b}) = ABC / \Delta_x$$

Pseudo-algorithm:

$$Dx = \text{Thresh}_x - \text{fabs}(\text{delta}_x); \quad Dy = \text{Thresh}_y - \text{fabs}(\text{delta}_y); \quad Dz = \text{Thresh}_z - \text{fabs}(\text{delta}_z);$$

$$\text{currentAngle} = \text{atan}(\text{delta}_z / \text{delta}_x);$$

$$\text{Anglesign} = \text{sign}(\text{currentAngle});$$

$$\text{higher} = (\text{sign}(\text{delta}_x * \text{delta}_z) + 1) / 2;$$

$$\text{WithinBox} = (\text{sign}(Dx) + 1) * (\text{sign}(Dy) + 1) * (\text{sign}(Dz) + 1) / 8;$$

if ($\text{Anglesign} > 0$)

$$\text{AngleSmallerThanMax} = (\text{sign}(\text{MaxAngle} - \text{currentAngle}) + 1) / 2;$$

else

$$\text{AngleSmallerThanMax} = 0;$$

$$\text{alltheconditionsTrue} = \text{higher} * \text{WithinBox} * \text{AngleSmallerThanMax};$$

if ($\text{delta}_x \neq 0$)

$$S = \text{alltheconditionsTrue} / \text{fabs}(\text{delta}_x);$$

else if ($\text{delta}_z \neq 0$)

$$S = \text{alltheconditionsTrue} / \text{fabs}(\text{delta}_z);$$

else

$$S = 0; \quad \text{return } S;$$

Table 3: Defining the similarity function S and the corresponding pseudo-algorithm are shown.

Since packet identification has predefined box dimensions, we define a box with its length along x , y and z coordinates. Then define the distance variables along each axis as D_x , D_y and D_z to check whether a given new feature (say \mathbf{b}) falls within the 3D box distance of the original feature (say \mathbf{a}). The A variable as defined in Table 3 checks for all these. Assuming that Δ_x is a nonzero value, the similarity measure is computed by the following equation: $S(\mathbf{b}, \mathbf{a}) = S(\mathbf{a}, \mathbf{b}) = ABC/\Delta_x$. This means that the closest features along the x axis will be more similar. (The complete pseudo-algorithm is given in Table 3). In the above equation, the variable A checks whether given two vectors fall within a predefined box of each other; B checks whether the features are elongated on the x - z plane in the ascending order, and C checks whether the angle between the given two vectors is less than 45° . If one of these conditions is not satisfied by the given two vectors, the final similarity value will become zero.

4.7. Results

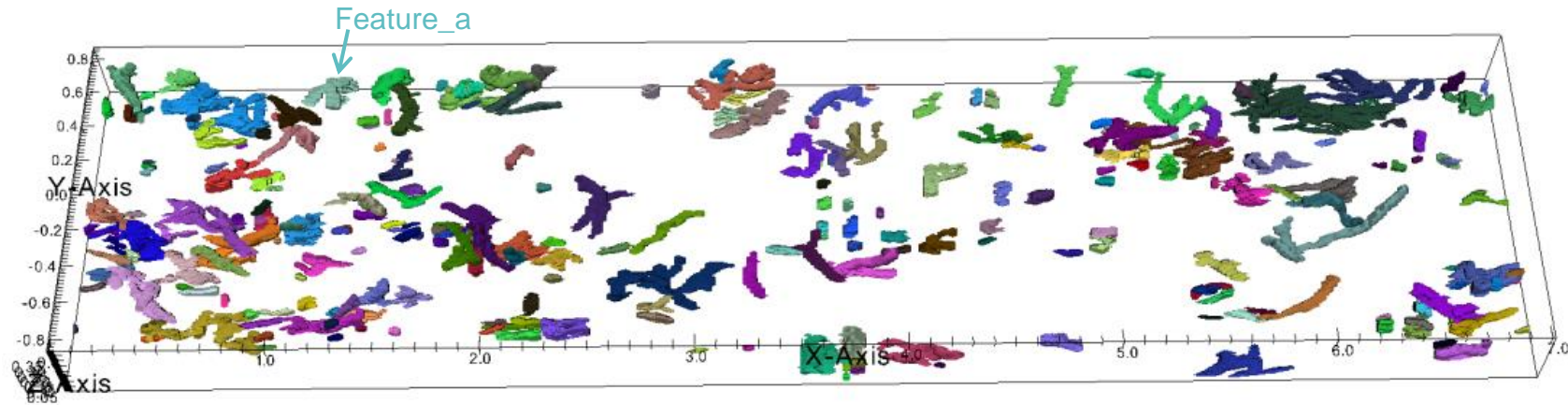
We apply our group tracking algorithm on the wall bounded turbulent data set described in Appendix-III. We implemented and used two types of grouping techniques in the results presented in this section. The first grouping technique is called “packet identification” and is a similar implementation of the presented algorithms in [53] and [65]. The second grouping technique is the clustering technique.

The described technique in [53] operates on each 2D plane in the 3D data individually to determine the groups. Our 3D implementation uses a node which has the maximum value within a feature. Thus, a feature is summarized with a

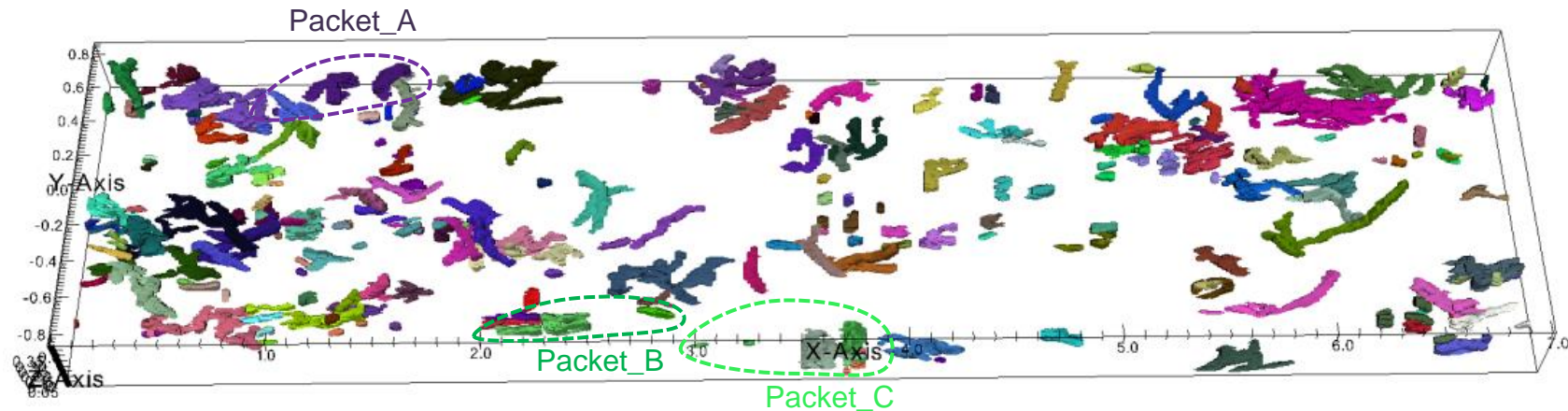
single node. This process uses the results of feature extraction and feature tracking. Therefore while the original algorithm is an $O(N)$ operation, where N is the number of total voxels in the data, our packet identification algorithm is an $O(M)$ algorithm where M is the number of the total features in the data. Therefore with the reasonable assumption where $M \ll N$, our algorithm works faster than the original technique, while the results are reasonably similar to the original algorithm.

The packet identification algorithm used in this dissertation can be summarized as:

- 1) Extract each feature and compute their attributes.
- 2) Represent each feature with its maximum swirling value and its location.
- 3) Take an ungrouped feature as the starting feature and give it a new and unique group ID.
- 4) Draw a bounding box around the maximum location of the starting feature and look for all the features that fall within the box.
- 5) For each feature within the box,
 - 6) Choose the closest unchecked feature (along the x axis) within the box and compute the angle between the maximum locations of these two features on (x,z) plane.
 - 7) If the angle is smaller than 45° , label the new feature with the same group ID and set it as the starting feature, then go to step 4.
- 8) Go to step 3.



(a) Feature tracking at t_1 . In feature tracking each **feature** has a unique color.



(b) Group tracking at t_1 . In group tracking each **group** has a unique color.

Figure 17: Visualization of feature tracking vs. group tracking in wall bounded turbulent flow simulation data. The original simulation data contains 46 timesteps. The variable being visualized is swirl magnitude; (a) Features extracted in t_1 where each feature has an automatically generated unique color (total 262 features); (b) Packets in t_1 where a total of 177 packets are identified and 3 sample packets (Packet_A, Packet_B and Packet_C) are circled. Packets are groups of features that travel together.

Figure 17a visualizes the extracted features in the first timestep of the wall bounded turbulent simulation data set. The individual features are extracted and visualized by using the feature tracking algorithm. In the figure, each feature has a unique color. Figure 17b visualizes the identified packets in the same timestep. The packets are formed of the extracted features in Figure 17a. In Figure 17b, each group has a unique color and the features that are the members of the same group have the same group color. Figure 18a visualizes the evolution of a selected feature Feature_a over the first five timesteps. Figure 18b visualizes the evolution of the entire group Packet_A (i.e., the packet) of Feature_a over the first five timesteps.

In the first timestep, the feature tracking algorithm extracted 262 features. These 262 features form total of 177 packets in the same timestep. Note that the total number of extracted features changes with the filtering parameters (thresholds). Similarly, the total number of identified packets changes with the group identification parameters. These parameters include the angle between the features and the bounding box size along x, y and z axis. The change in total number of extracted features versus the total number of identified packets in each timestep can be seen in Figure 19. The results shown in Figure 19 indicate that a packet is formed of two individual features on average.

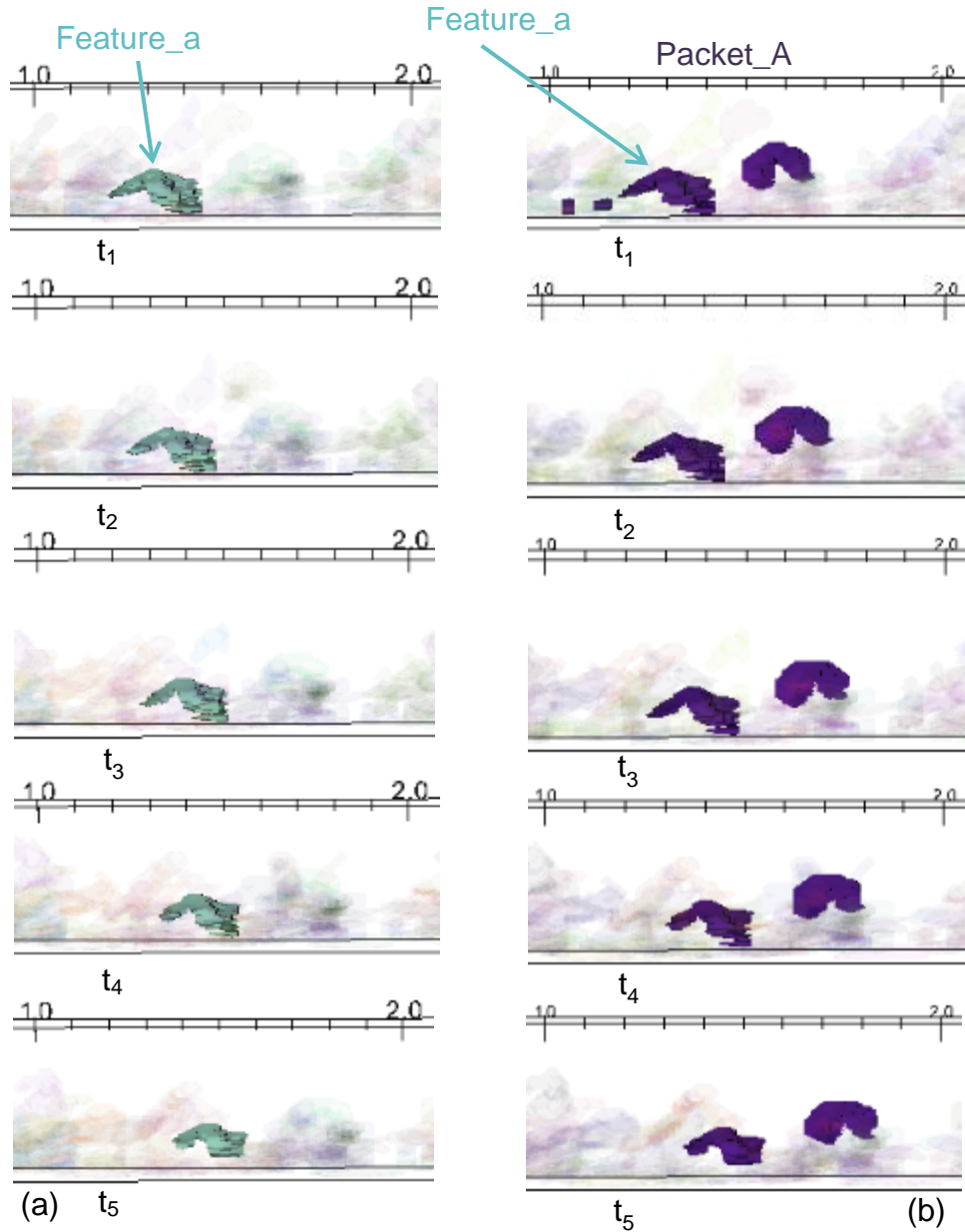


Figure 18: The evolution of a selected feature and its group is visualized in the first five timesteps. (a) The evolution of a single feature (Feature_a from Figure 17) is visualized in the first five timesteps. All the features are extracted and then tracked by feature tracking algorithm. Except Feature_a, all the other features are visualized transparently. (b) The evolution of Feature_a's group (Packet_A) is visualized in the first five timesteps. Groups are determined (i.e., extracted) and tracked by using group tracking algorithm. Except Packet_A, where Feature_a \in Packet_A, all the other groups are visualized transparently. Features join and leave packets throughout a packet evolution.

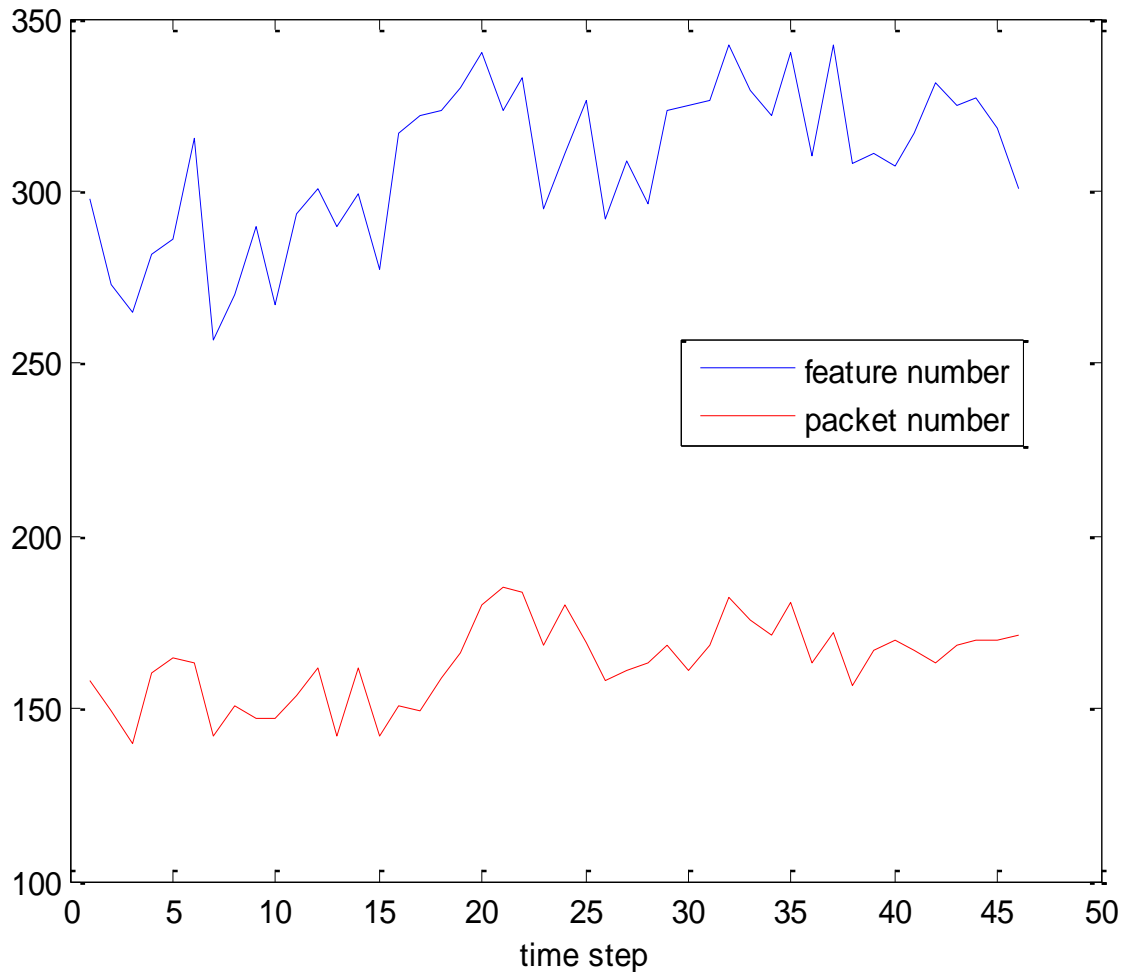


Figure 19: The total number of extracted features vs. the total number of identified packets in each timestep. The data set contains 46 timesteps. The blue color shows the total number of features in each timestep. The red line shows the total number of identified groups in each timestep.

Group tracking algorithm is applied on the data set with the defined similarity function (see Table 3) and the clustering algorithm. The results are summarized in Figure 20 and Figure 21. Figure 20 visualises the sample time steps 1, 2 and time step 8. Figure 21 shows the results in time steps 21, 26 and 31. In the figures, each group has a unique color and the same packet gets the same color

in the next time step. These results were obtained with the parameters where $Thresh_x=0.3$, $Thresh_y=0.08$ and $Thresh_z=1$. The features with a volume that is smaller than 25 are filtered out. Therefore, such small features do not show up in the figures.

Notice that as the grouping parameters change, the size of a packet and the total number of features in each packet change. The number of features change based on the segmentation parameters (the threshold value used in the region growing case). For the grouping, if the segmentation parameters remain the same, then the grouping becomes allocating these segmented features into scientifically meaningful groups. Therefore, the total number of groups decreases as the grouping parameters increase in our results.

Group extraction and tracking has also been applied on the larger simulation which is described in Appendix III. The resolution of this data is $2520 \times 1120 \times 110$ and it contains 250 timesteps. A sample result of applying the group tracking algorithm on this data set is shown in Figure 22. Figure 22a visualizes all the extracted features and their groups in the first timestep. In the figure, each group has a unique color. There are 3828 features visualized in this figure. These 3828 features are grouped into 3244 packets. Figure 22b visualizes a selected area in Figure 22a to magnify the details. It would require magnifying the figure several times to see the features clearly. Figure 22c visualizes sample features in detail after many magnifying process.

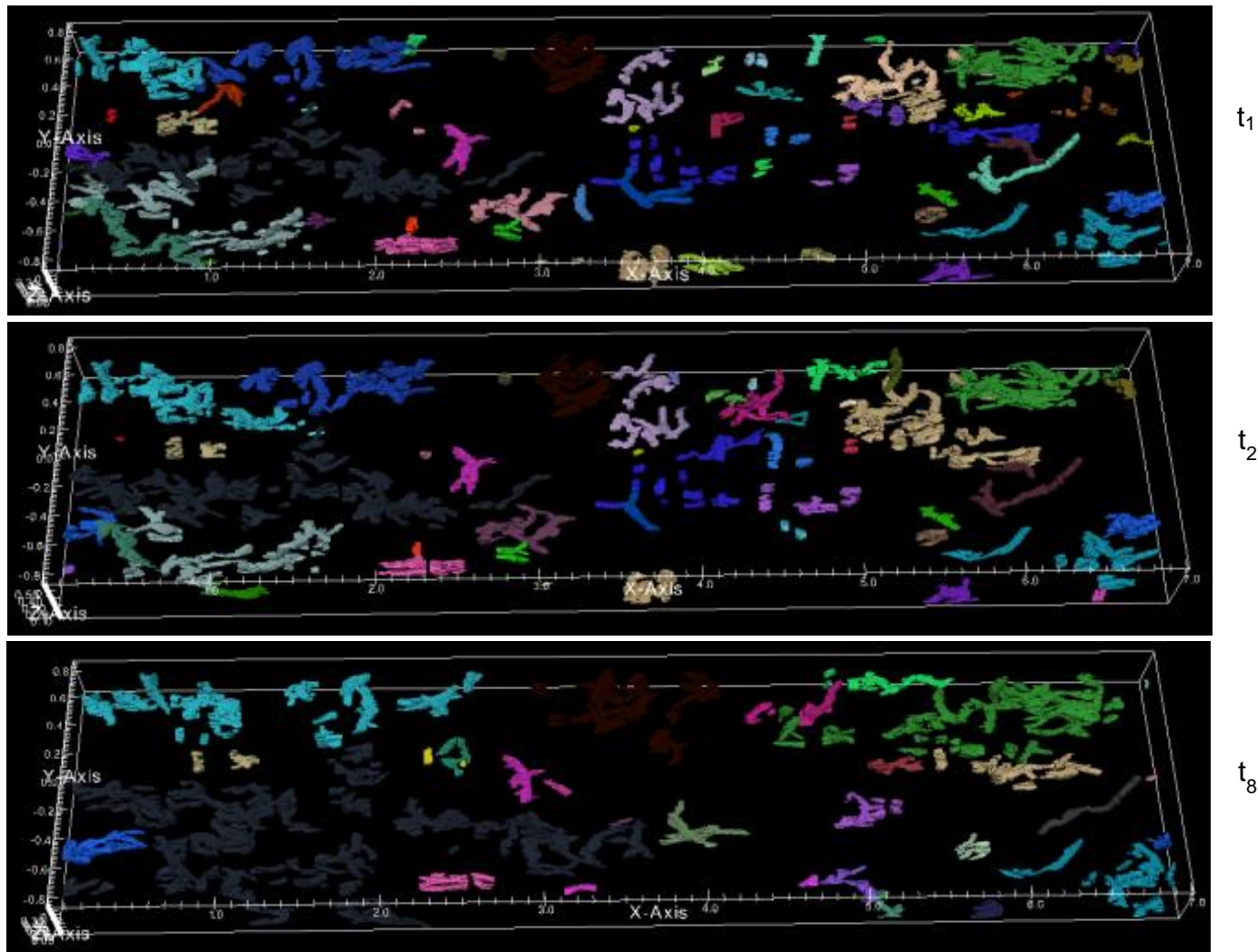


Figure 20: Group tracking results are visualized in timesteps 1, 2 and in timestep 8. The visualized data is wall-bounded turbulence simulation.

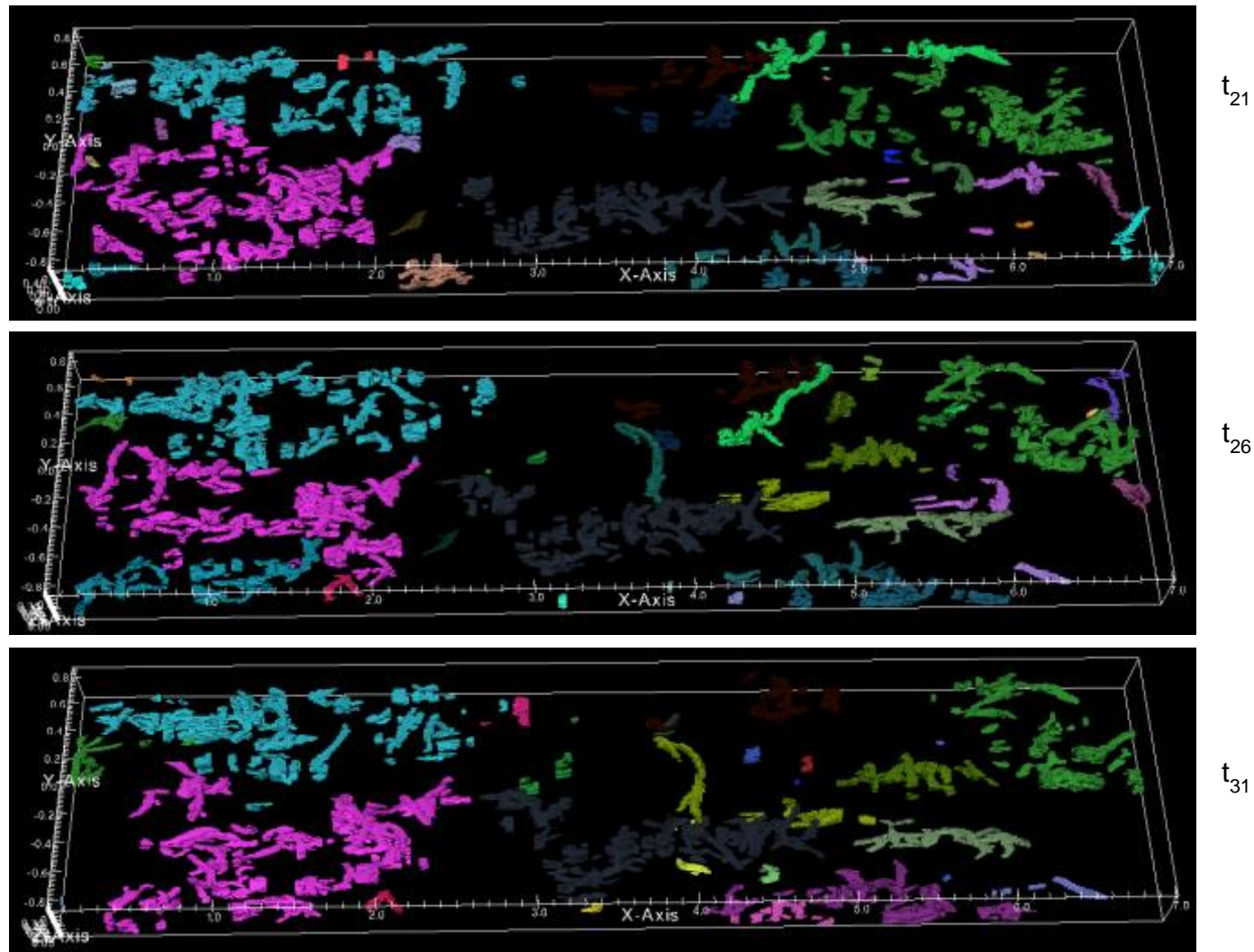


Figure 21: Group tracking results are visualized in timesteps 21, 26 and in timestep 31. The visualized data is wall-bounded turbulence simulation.

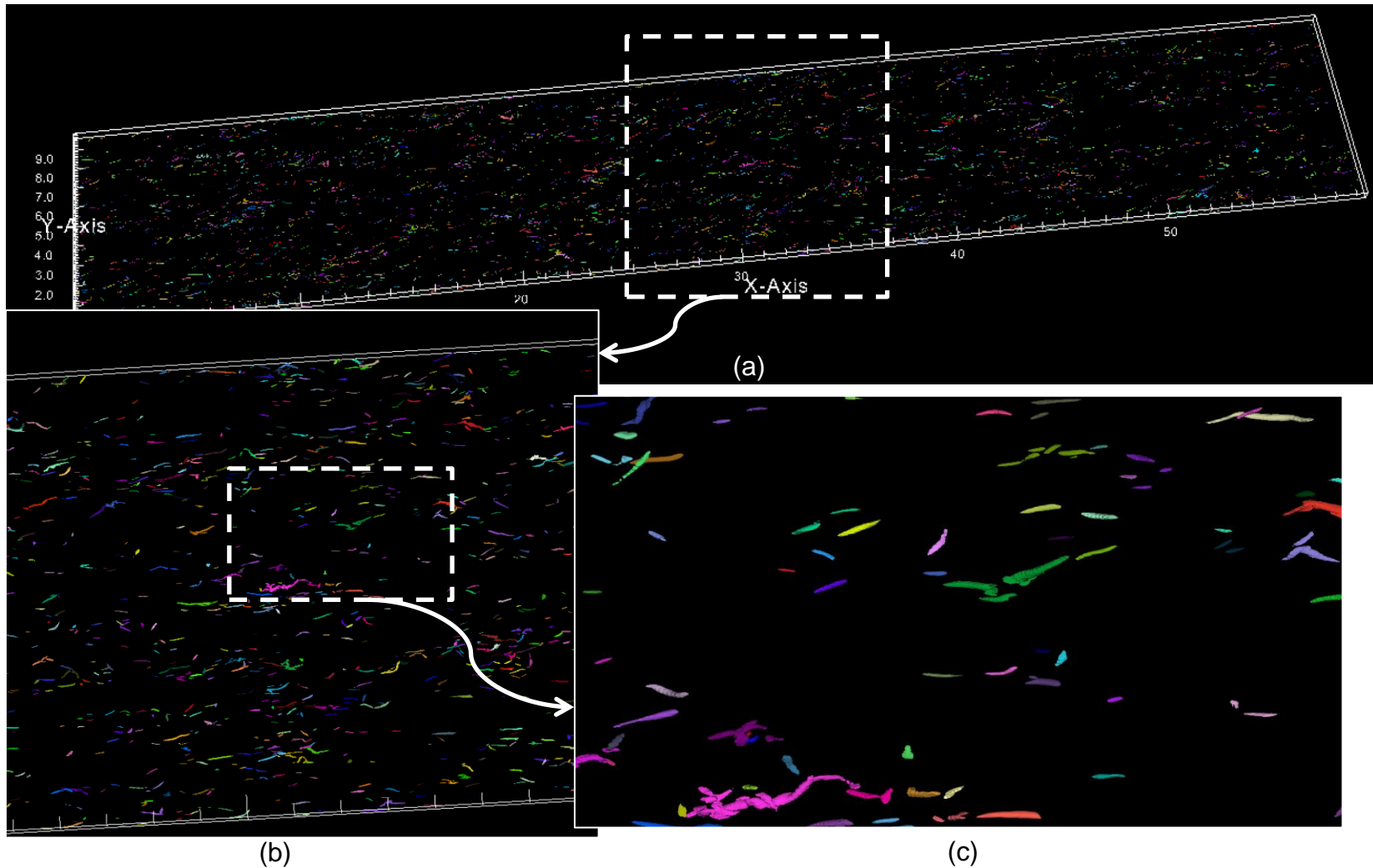


Figure 22: Group extraction is shown on the large data set. (a) in this first timestep, there are 3828 features are extracted. All these 3828 features are grouped into 3244 packets and these packets are visualized. (b) A selected area is magnified. (c) an area within the selected area is further magnified to see individual features' shape.

CHAPTER 5:

PETRI NETS FOR ACTIVITY DETECTION

The easiest way to model an activity is using a flowchart or a state graph. Petri Nets are one such formalism supporting parallel and distributed systems. Therefore, in this dissertation, Petri Nets are proposed to model and detect activities. This chapter, first describes the fundamental Petri Net formalisms and then discusses the existing and unaddressed problems in the existing activity detection (recognition) applications using Petri Nets.

In this dissertation, we propose using Petri Nets for activity modeling, activity detection and for hypothesis validation. In this chapter, we provide the essential background for Petri Nets. Then we will discuss the existing and unaddressed problems in the existing activity detection applications with Petri Nets.

Petri Nets were introduced by Carl Adam Petri in [60]. Essentially, Petri Nets are graph based techniques that can model parallel and distributed systems and are used in many fields including software development, workflow management and manufacturing. Since their initial introduction, due to new requirements of various fields and domains, the capability of Petri Nets increased yielding various Petri Net formalisms such as *Marked Petri Nets*, where places need to include more than one token; *Timed Petri Nets*, where a token needs to wait a certain time (delay) to become available for enabling a transition; *Probabilistic Petri Nets*,

where the delay of Timed Petri Nets varies by including uncertainty and *Coloured Petri Nets*, where IDs are used to distinguish between tokens. The detailed information about these different Petri Net formalisms can be found in the books [59] and [20]. Petri Nets are also used in software engineering applications. As a result, there have been a significant amount of Petri Nets work has been proposed related to software development. An example of such work is *Object Petri Nets* support polymorphism, inheritance and dynamic binding [37]. Note that, later on the term “Object Petri Nets” is also used in activity detection applications indicating that the tokens are objects in a Petri Net [25], [40].

Petri Nets can help modeling (describing) activities. In order to do that, an activity is analyzed semantically and then is decomposed it into its key points. These key points construct the nodes in the time sequence, i.e. the Petri Net model. Such key points include feature (actor) states, and atomic (primitive) events (i.e., actions). Petri Nets (PN) are one of many techniques that allow graphical modeling of discrete systems and complex temporal events [52].

5.1. Fundamental Petri Net concepts and components

In general, all the Petri Net formalisms include four main components. These are: (1) the set of places **P**, (2) the set of transitions **T**, (3) the input function **I** that maps a transition to a set of places, (4) the output function **O** that maps a place to a set of transitions. Therefore a generic Petri Net structure is a four-tuple (**P**, **T**, **I**, **O**) where:

- $\mathbf{P} = \{p_1, p_2, p_3, \dots, p_n\}$ is the set of n places which are drawn as circles in the graph,
- $\mathbf{T} = \{T_1, T_2, T_3, \dots, T_k\}$ is the set of k transitions which are drawn as bars (rectangles) in the graph,
- $\mathbf{I}_{n \times k}$ is the input transition relation matrix between the places and transitions where the relation is usually defined in terms of arcs such that $\mathbf{I}(j, i) = k$ is the arc weight from j^{th} place to the i^{th} transition. We will call this type of arcs (from a place to a transition) as incoming arcs.
- $\mathbf{O}_{n \times k}$ is the output transition relation between the transitions and places where the relation is usually defined in terms of arcs such that $\mathbf{O}(j, i) = k$ is the arc weight from i^{th} transition to the j^{th} place. We will refer to this type of arcs (from a transition to a place) as outgoing arcs.

The arc weights are usually represented on each individual arc. The arcs without any specific weight number are assumed to have the arc weight 1. In a classical Petri Net graph, a transition cannot have an edge (connection) to another transition and similarly a place cannot have an edge to another place. In the literature, places can also be classified further as being preconditions or post-conditions [59], [20].

An example of a Petri Net is shown in in Figure 23. The Petri Net models the left bag (unattended bag) activity that is previously described in Figure 4 in Chapter 1.

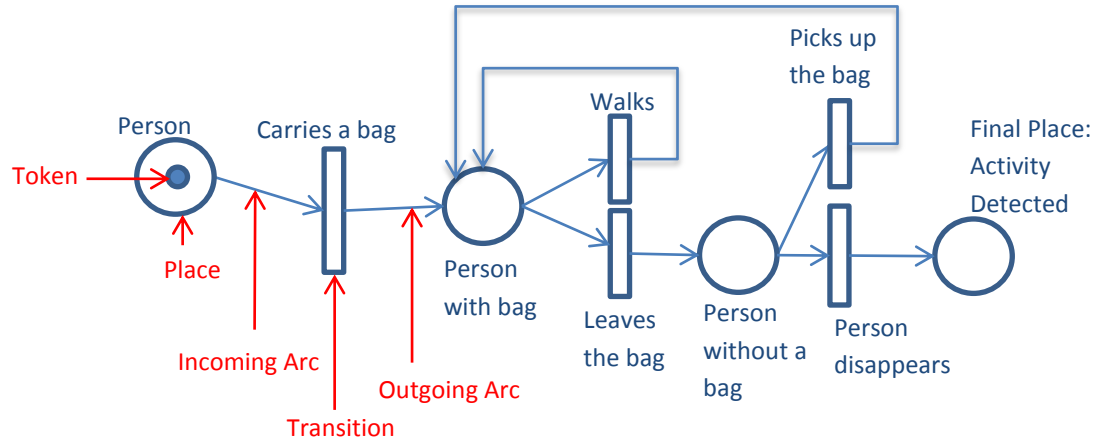


Figure 23: Illustration of Petri Net components on the Petri Net model of the unattended bag example.

5.2. Marked Petri Nets

Marked Petri Nets allow to include multiple “tokens” to be in each place. A token is the primitive structure that a Petri Net operates on and is represented by a solid dot in a graph (see Figure 23). A deterministic and marked Petri Net is formed of places, transitions, arcs and tokens, therefore is a 5-tuple $(\mathbf{P}, \mathbf{T}, \mathbf{I}, \mathbf{O}, \mathbf{M})$ in which:

- $\mathbf{M} = \{m_1, m_2, m_3, \dots, m_n\}$ is the current marking summarizing the configuration of the tokens in a given Petri Net where m_i is the number of tokens at the place p_i .

The functions \mathbf{I} and \mathbf{O} represent the set of arcs between places and transitions. The state of a marked Petri Net is defined by the number of tokens in each place at a certain timestep. If it is not specified on the graph, then the weight of each arc is assumed to be 1.

The dynamics of a marked Petri Net is defined by two operations: *enabling* and *firing* a transition. A transition T_i is *enabled* in \mathbf{T} if and only if there are enough tokens in each input places for the consumptions to be possible:

$$\forall j : \mathbf{M}(j) \geq \mathbf{I}(j, i) \quad (3)$$

Firing a transition T consumes $\mathbf{I}(i, j)$ tokens from each of its input place i , and produces $\mathbf{O}(j, i)$ tokens in each of its output places j . As such, the final marking of a marked Petri Net can be computed with the following equation:

$$\mathbf{M}_k = \mathbf{M}_{k-1} + \mathbf{K} \cdot \mathbf{E}_k \quad (4)$$

where $\mathbf{K} = (\mathbf{O} - \mathbf{I})$ and \mathbf{E}_k is the vector of the transitions that are fired. Therefore $\mathbf{E}(i) = 1$ if T_i is fired, and 0 otherwise.

Figure 24 shows an example Petri Net constructed with 4 places and 2 transitions where $\mathbf{P} = \{p_1, p_2, p_3, p_4\}$, $\mathbf{T} = \{t_1, t_2\}$, $\mathbf{M} = \{2, 0, 0, 0\}$,

$$\mathbf{I} = \begin{bmatrix} p_1 & t_1 & t_2 \\ p_2 & 2 & 0 \\ p_3 & 1 & 0 \\ p_4 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{O} = \begin{bmatrix} p_1 & t_1 & t_2 \\ p_2 & 0 & 0 \\ p_3 & 1 & 0 \\ p_4 & 0 & 1 \end{bmatrix}, \text{ therefore } \mathbf{K} = \mathbf{O} - \mathbf{I} = \begin{bmatrix} p_1 & t_1 & t_2 \\ p_2 & -2 & 0 \\ p_3 & -1 & 0 \\ p_4 & 1 & 1 \end{bmatrix},$$

At the timestep t_1 there are only 2 tokens in place p_1 . Therefore nothing happens in this case. Now, assume that a new token enters to p_2 at timestep t_2 (as shown in Figure 24b). The new marking \mathbf{M} becomes $\mathbf{M} = \{2, 1, 0, 0\}$. At this moment, the condition (1) is satisfied for T_1 since $\mathbf{M}(1) = 2$ and $\mathbf{M}(2) = 1$. The firing process moves the tokens from p_1 and p_2 to p_3 . The final marking can be computed as:

$$\mathbf{M}_2 = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 0 \\ -1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad (5)$$

However, notice that in this case, firing T_1 enables the transition T_2 . Therefore in this case, the final marking in timestep t_2 becomes:

$$\mathbf{M}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 0 \\ -1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (6)$$

Once T_1 is fired, the new marking enables T_2 . Firing T_2 gives the final marking of the Petri Net in timestep t_2 .

5.3. Timed and stochastic Petri Nets

Timed Petri Nets add time constraint to places (or transitions) by allowing time dependent operations in marked Petri Nets. Therefore it is a 6 tuple $(\mathbf{P}, \mathbf{T}, \mathbf{I}, \mathbf{O}, \mathbf{M}, \mathbf{D})$ where $\mathbf{D} = \{D_1, D_2, \dots, D_n\}$. D_i is the timing associated with place p_i (or transition T_i). When a token falls into the place p_i , it becomes unavailable for the duration D_i . This token, therefore cannot be moved to another place before the time D_i has elapsed. It becomes available after the duration D_i passed again. This kind of schema is especially useful to limit the movement of tokens in a Petri Net by time. For example, consider Figure 24d. In timestep t_2 , the token does not stay in p_3 since it immediately moves into the place p_4 . A time constraint in p_3 would avoid this case, by forcing the token to remain in this place for a pre-specified D_3 time duration. At any time, the final marking \mathbf{M} is the sum of \mathbf{M}^a and

\mathbf{M}^u where the marking \mathbf{M}^a is made up of the available tokens and the marking \mathbf{M}^u is made up of the unavailable tokens.

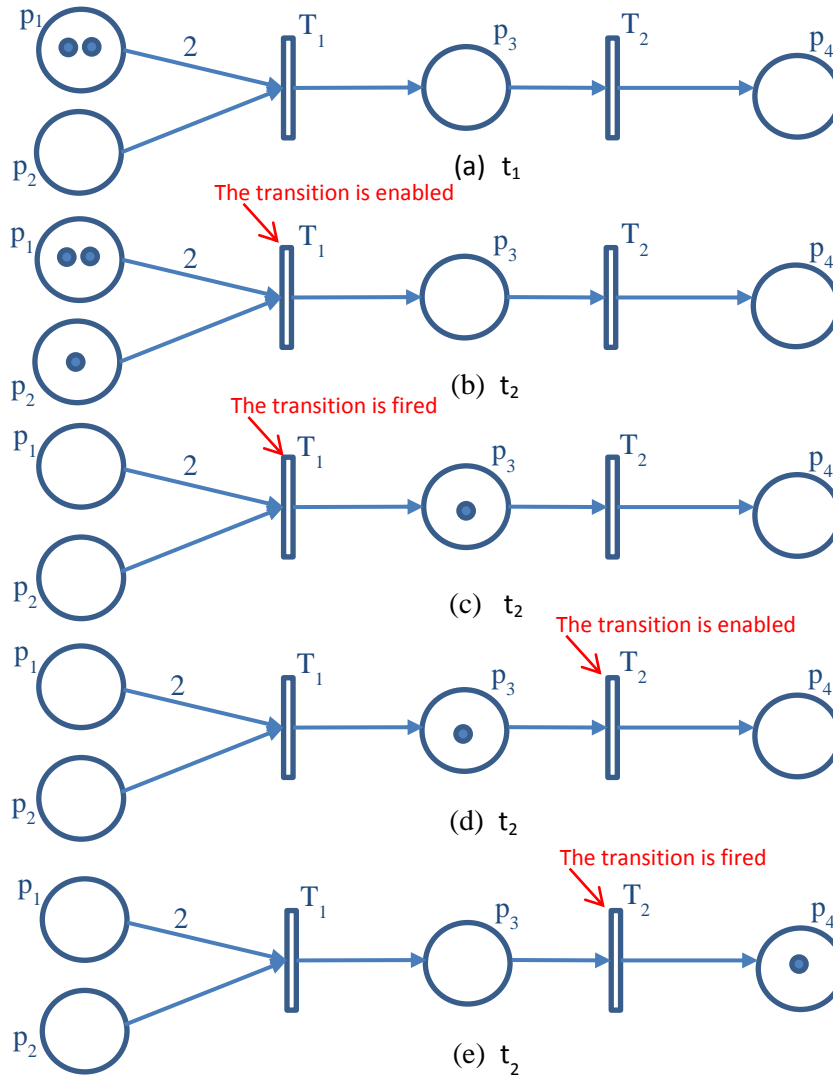


Figure 24: An example of enabling and firing a transition in a marked Petri Net. When a transition is fired, tokens move along the Petri Net.

In timed Petri Nets, the time duration associated with each place (or each transition) is a fixed value. However, in some cases, this duration varies. To include such variance in durations in timed Petri Nets, stochastic Petri Nets have

been proposed. In stochastic Petri Nets, each token is assigned a probabilistic duration in each place [59], [20].

5.4. Coloured Petri Nets

In a Marked Petri Net, all the tokens are considered identical. However, in some applications (as in activity detection applications), tokens need specific identifiers (ID) attached. To handle such situations Coloured Petri Nets have been proposed. In a Coloured Petri Net, a color (i.e., an ID) is attached to each token [20]. Coloured Petri Nets also allow attaching a function to each arc. Such an arc function can map a color (an ID) to another one or they can be conditioned based on the color. For instance, an arc can transform the color yellow into the color blue, and the color red to the color green. The tuple **(P, T, Pre, Post, M₀, C)** represents a Coloured Petri Net where **Pre** is the set of functions associated with incoming arcs and **Post** is the set of functions associated with the outgoing arcs. **C** = {C₁, C₂, ...} is the set of colors and the color C_k is the n tuple i.e. C_k = <C_{k1}, C_{k2}, C_{k3}, ..., C_{kn}>. For example, in a typical activity detection application, C_{kn} may represent an object's attribute (such as volume, mass, shape, etc.). Therefore we can use colors to encode attributes.

5.5. Earlier definitions of the enabling and firing processes

The dynamics of typical Petri Nets are defined by two fundamental operations, namely *enabling* and *firing* a transition. *Enabling* a transition fundamentally indicates that the tokens “can” move from the input places to the output places of the transition. *Firing* is “the process” of moving the tokens from

those input places to the output places. The exact definitions of these two terms change slightly from application to application. For example, according to the work in [25], a transition is enabled if all the input places have at least one token. Firing a transition removes one token from every input place and inserts a token to every output place.

A similar firing and enabling schema is defined by Lavee et. al. in [40]. In their work, they state that a transition is *enabled* when all the input places have tokens at least the number of arc weight. In their work, they also state that the enabling rule can be modified to contain conditions on the properties of tokens. *Firing* a transition removes a number of tokens from each of the transition's input places equal to the arc weight and creates a number of new tokens in each of the output places equal to the arc weight [40]. Assigning a priority to each transition could solve the conflict problem. A timed transition fires only if its duration parameter is greater than the time condition of the place (or of the transition).

Albenese et. al. in [3] state that a transition *fires* if and only if all its input places have a token and only if the transition condition is satisfied. When a transition *fires*, all enabling tokens are removed and a token is placed in each of the output places.

Similar enabling and firing processes are also defined by Perse et. al [58]. In their work, a transition T_j is *enabled* if and only if $M(p_i) \geq l(p_i, T_j)$ where $M(p_i)$ is the number of tokens in the input place p_i .

5.6. Unaddressed problems for activity detection

As mentioned previously, most computer vision applications use objects as tokens, places as object states and transitions as actions or conditions. There are several problems associated with the existing Petri Net applications in activity detection for scientific visualization. These problems are summarized below:

- **Handling the change in an object's state or an object's attributes:**

In earlier system applications of Petri Nets (examples can be found in [37] and [52]), tokens usually are expected to remain in a place and do not split, merge, disappear or change their shapes (or their states) in a place over the time. However this is not the case in computer vision or in scientific simulation, especially when a place represents an object's state and a transition represents an object's action. Objects dynamically change their state or their attributes from one timestep to the next and this change somehow has to be considered within a Petri Net formalism. For example, consider a case where a place represents "feature with volume>50". In this example, a feature may fall into this place if its volume is greater than 50 nodes (voxels) in a timestep. Assume that this feature's volume becomes 30 since it shrinks in the next timestep. In this case, it should no longer remain in the place in the next timestep even if there is no transition is designed to fire the token (i.e. the feature) for this case. Therefore, only the tokens that are currently still in the same object state should remain in the same place. The natural way of doing this is including the change, i.e., time variance in tokens, into the Petri Net formalism.

In recent activity detection applications with Petri Nets, a place represents an object's state. An object's state can be any set of quantitative attributes helping to describe the activity. For example, an object's state can be "a person without the bag" while in another place, a state could just be "a person" as in Figure 23. In such situations (in the literature this type of Petri Nets also called Object Petri Nets as in [40]) the existing applications do not check whether a token still remains in the same place which represents an object state. In cases where a token no longer satisfies a place's definition, it still remains in the same place until it is fired. However, if the situation describing the state change is not modeled in the given Petri Net, then the token is simply "stuck" in its current place.

- **Handling the change in token numbers due to the merge and split events:**

In scientific simulations, the number of features may change from one timestep to the next. In Petri Nets, where a token represents a feature with its attributes, the existing tokens need to be updated to reflect this change. Scientific dynamics include merge, split, continue and disappear events [72] and these events model the change in the total number of features from one timestep to the next. The number of actors for split and merge events may change from one instance to another. For example, in one merge instance two features may merge to form one and in another merge instance five features may merge to form a single feature. Similarly, in one split instance, one feature may split into four features while in another split case one feature may split into two. This

variability in the number of merging and splitting features makes it harder for the scientist to model these events even with the arc functions in Coloured Petri Nets. Moreover, even if these events are not explicitly modeled in a given Petri Net, the Petri Net should still consider the fundamental dynamics of the scientific environment. Therefore, in the next chapter we propose a new Petri Net formalism that handles these above-mentioned problems.

CHAPTER 6:

TOKEN-TRACKING PETRI NETS FOR ACTIVITY

DETECTION

Petri Nets do not consider the dynamics of scientific simulations.

Therefore in this chapter, we present token-tracking Petri Nets to handle the problems discussed in the previous chapter. Token-tracking Petri Nets can handle the time variance in tokens and support the dynamics of scientific environments by incorporating the tracking information.

In all the earlier Petri Net applications of activity detection, the tokens are usually expected to remain in a place and do not split, merge, disappear or change their shapes, attributes or states in a place over the time until they are fired. However, this is not the case in many applications. Objects (thus the tokens) dynamically change their state or their attributes from one timestep to the next and this change somehow has to be included within a Petri Net formalism. , This change is considered and included by coupling the Petri Net with the tracking information in the proposed enhanced Petri Net. We enhance Petri Nets and call the enhanced version *token-tracking Petri Nets*. Token-tracking Petri Nets (TTPN) consider the feature dynamics by updating the tokens and their places automatically as the time changes. In TTPN, this change is considered and included by coupling the Petri Net with the tracking information.

A TTPN is a 10 tuple $(\mathbf{P}, \mathbf{T}, \mathbf{I}, \mathbf{O}, \mathbf{C}_P, \mathbf{C}_T, \mathbf{S}, \mathbf{E}, \mathbf{M}_{k-1}^+, \mathbf{F})$ where \mathbf{M}_{k-1}^+ is the final marking (see Figure 25) obtained in the previous timestep t_{k-1} and \mathbf{F} is the *updating function* that maps the existing tokens in a Petri Net from t_{k-1} to the tokens extracted in t_k such that:

$$\mathbf{F}(\mathbf{M}_{k-1}^+) = \mathbf{M}_k^- \quad (7)$$

The state of a TTPN (i.e., the configuration of tokens in places) is a function of time and is described by the tuple $(\mathbf{M}_{k-1}^+, \mathbf{F})$. in Petri Nets. \mathbf{M}_{k-1}^+ represents the final marking of the Petri Net in timestep $k-1$ and \mathbf{M}_k^- represents the initial marking in timestep k . While the existing Petri Net applications assume that $\mathbf{M}_{k-1}^+ = \mathbf{M}_k^-$ (as illustrated in Figure 25-Case A), this assumption does not hold in scientific simulations (as illustrated in Case B, Case C and Case D) since tokens become time dependant. Therefore, in general we can say that $\mathbf{M}_{k-1}^+ \neq \mathbf{M}_k^-$. TTPN is designed to handle such situations in scientific simulations.

Similar to coloured Petri Nets, each token has an ID in TTPN. The marking $\mathbf{M} = \{\mu_1, \mu_2, \mu_3, \dots, \mu_n\}$ summarizes the distribution of tokens in a given Petri Net. μ_n is the set of tokens in place P_n such that $\mu_n = \{\mathbf{X}_1, \mathbf{X}_2, \dots\}$. A token \mathbf{X}_a is n tuple such that $\mathbf{X}_a = (x_{a1}(t_k), x_{a2}(t_k), x_{a3}(t_k), \dots, x_{an}(t_k))$ where $x_{an}(t_k)$ is the n^{th} attribute of the token \mathbf{X}_a at timestep t_k . This means that the tokens in TTPN are time dependent. The initial marking \mathbf{M}_k^- of a Petri Net is the marking that is mapped from the previous timestep t_{k-1} to the current timestep t_k and the final marking \mathbf{M}_k^+ is the marking where all the enabled transitions has fired such that no further enabled transition remains in the timestep t_k . The mapping from previous timestep to the current one is done by use of a function. Tracking information is

used to update the object attributes in this dissertation. This process involves extracting new tokens in the data set and then assigning them to the existing tokens from the previous timestep in the Petri Net by use of tracking information. We call this process the update process of TTPN.

Once all the tokens and their places are *updated* via the function \mathbf{F} , the next step is evaluating the Petri Net by firing all the enabled transitions for each token. Similar to typical Petri Nets, firing is done by rewriting Equation (1) for a given token \mathbf{X}_a such that

$$^{Xa}\mathbf{M}_k^+ = ^{Xa}\mathbf{M}_k^- + (\mathbf{O} - \mathbf{I}) \cdot \mathbf{E}_k \quad (8)$$

where $^{Xa}\mathbf{M}_k^+$ is the new location of the token \mathbf{X}_a in the Petri Net. The same token needs to be in the all input places to enable a transition (along with the transition condition). In a given TTPN model, each arch weight is considered 1 to model a hypothesis or an activity. Furthermore, TTPN considers the dynamics of the system including merge, split, appear, disappear and continuation internally and therefore, a scientist does not need to model these events at each place explicitly. This process incorporates the time variance in Petri Nets and simplifies the modelling of an activity. An overview is given in Figure 26. Consider the given Petri Net model with its tokens in timestep t_{k-1} with its existing tokens in Figure 26a. The green token (e.g., a feature) in P_2 splits into three tokens (e.g., three features) in timestep t_k (shown in Figure 26b). Traditional Petri Nets do not allow a token split while waiting in the same place. However, the TTPN can do this by using sub-nets. A sub-net is a Petri Net in which the time information is attached to both tokens and the arcs.

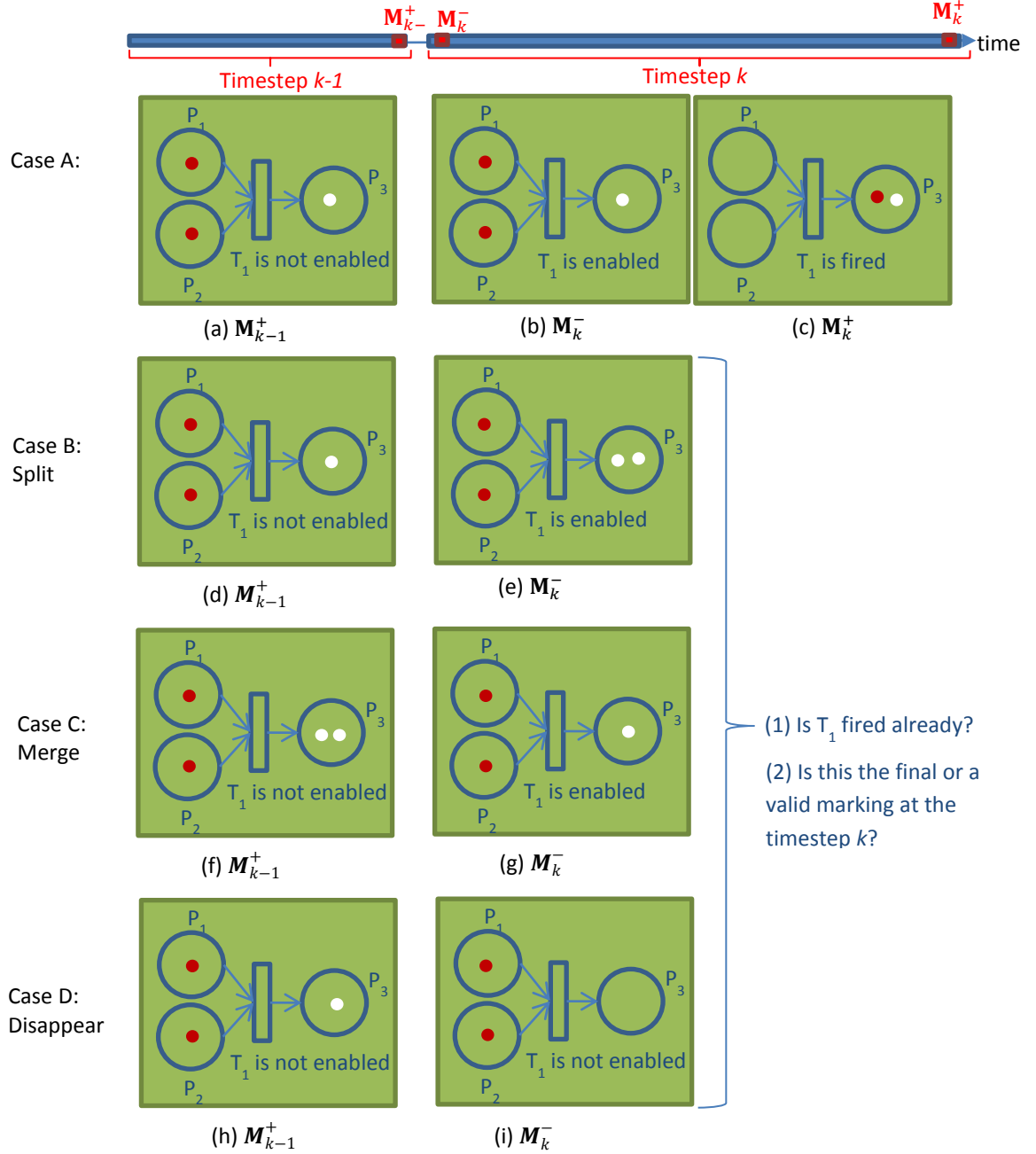


Figure 25: Various illustrations of different time dependent problems in Petri Nets. M_{k-1}^+ represents the final marking of the Petri Net in timestep $k-1$ and M_k^- represents the initial marking in timestep k . While the existing Petri Net applications assume that $M_{k-1}^+ = M_k^-$ (as illustrated in Case A), this assumption does not hold in scientific simulations (as illustrated in Case B, Case C and Case D) since tokens become time dependant therefore in general that $M_{k-1}^+ \neq M_k^-$. TTPN is designed to handle such situations in scientific simulations.

In TTPN, each place with its tokens (except the final place) is first isolated from the given Petri Net model and converted into a sub-net as shown in Figure 26c. The main purpose of the sub-net is to correlate the existing tokens in a given Petri Net from t_{k-1} to the new tokens extracted in t_k by using the tracking information. Therefore the sub-net is a time-dependent Petri Net.

For each place P_i , the sub-net creates two additional (pseudo) places P_i'' and P_i' (shown with black circles in Figure 26). Correlating the tokens from t_{k-1} to the extracted ones in t_k is graphically represented by the combination of a black arc, a transition and a red arc. The red arcs are defined only for the tokens from t_k and the black arcs are defined only for the tokens from t_{k-1} . From one timestep to the next, a scientific feature will either merge, split, disappear or continue. The transitions *merge*, *splits*, *disappears* and *continues* are obtained from the tracking information. For the *merge* transition, the sub-net removes b merging tokens from t_{k-1} and puts the merged token from t_k into P_i'' . Similarly, the *splits* transition moves the splitting token from t_{k-1} in P_i and puts a number of corresponding tokens from t_k into P_i'' . The values of the variables a and b (along with the token IDs) are obtained from the tracking information. The *continues* transition removes a token from t_{k-1} in P_i and puts the corresponding token from t_k into P_i'' . The *disappears* transition removes disappearing tokens from P_i to P_i' .

Once the sub-net reaches its final marking, the tokens remaining in P_i'' are the ones that changed their state during the transition from t_{k-1} to t_k , and the tokens in P_i' are the disappearing ones during the transition from t_{k-1} to t_k .

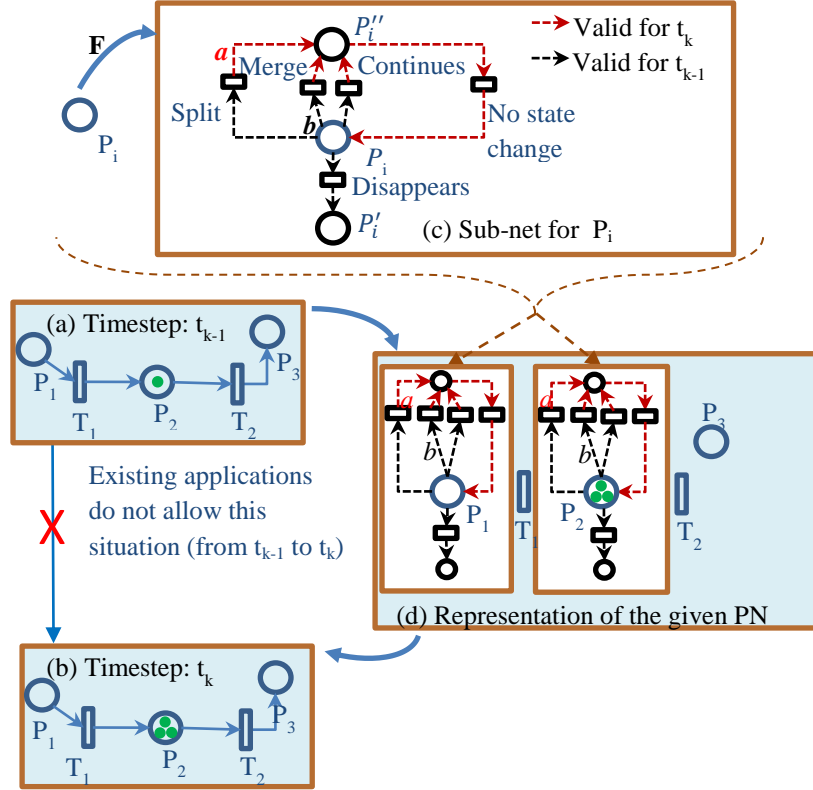


Figure 26: The use of sub-nets in TTPN. (a) a scientist given Petri Net and its marking at the timestep t_{k-1} is shown, (b) the marking of the Petri Net (PN) changes in t_k since the token splits into 3 objects. However since there is no transition is fired, the marking cannot change in traditional Petri Nets; (c) to solve this, each place is represented with a sub Petri Net (sub-net) in TTPN. The sub-net can model and allow the change in the marking of PN during the transition from t_{k-1} to t_k . (d) Each place (except the final place) in the PN is replaced with this sub-net and once the execution of the sub-net is completed for each place, the update process is completed and the initial marking is obtained in t_k .

Depending on the domain, the tokens in both places P_i' and P_i'' can be moved back into the place P_i , can be discarded or can be moved back into one of the initial places. Figure 26d illustrates the update process for the Petri Net shown in Figure 26a. In Figure 26d, each place (except the final place) is converted into a sub-net, then each of these sub-nets is executed independently. The update process replaces the single token in Figure 26a with 3 tokens. Once the update process is completed, the isolated places with their updated tokens

are put back into the given model. Therefore while the direct transition from Figure 26a to Figure 26b is not defined in typical Petri Nets, this transition becomes possible through the TTPN (by defining and using sub-nets).

After running the update process, the new (updated) tokens can be used to execute the given model to obtain the final marking in t_k . This is done by using Equation 8.

6.1. Activity detection framework with TTPN

Figure 27 shows how the framework operates in each timestep. The input to the system is the time-varying data set and the Petri Net model which is defined by the scientist (see next section). The first step is processing the data in the framework. In this step, features, groups, variable changes or other types of user interested entities are computed. Different types of features can be extracted by using appropriate tools for the respective domain. The computed meta-data may include volume, mass, centroid, max and min locations, max and min positions, orientation, shape information, etc. Once all the tokens are formed in t_0 , they are used to execute the Petri Net starting from the initial place. Both the meta-data and the final marking \mathbf{M}_0^+ is passed into the next timestep t_1 .

In timestep t_1 , first the data at t_1 is processed to extract features and groups. Then their meta-data is computed. This meta-data is transformed into tokens. Next step is correlating the extracted features and groups to the extracted ones in t_0 . Any of the available tracking algorithms (such as volume overlap, prediction or time-varying contour based algorithms) can be used to correlate features and groups or it may be inherent in the simulation. The tracking step computes

various attributes including the tracking history of the features (correspondence list), position changes, and any other value/attribute that is a function of two consecutive timesteps. Both the newly formed (extracted) tokens and computed tracking information are fed into the Petri Net for activity detection. In the Petri Net, the first step is correlating the existing tokens from t_0 to the tokens extracted in t_1 . Once the Petri Net is updated by using the tracking information, the marking \mathbf{M}_1^- is obtained. Then, the Petri Net is executed to obtain the final marking \mathbf{M}_1^+ .

Both the computed meta-data at t_1 and \mathbf{M}_1^+ are fed into the next step. This process repeats itself recursively for each timestep. The meta-data that comes from the previous timestep is updated with the new tokens extracted in the current timestep by using the tracking information in each new timestep. Tokens which fall into the final place are the ones performing the complete activity.

When the evaluation over time is completed, the list of tokens with their token histories in the final place can be used to generate an activity list. This list, can be used for visualization and further data analysis purposes.

6.2. Modeling with Petri Nets

The scientist can model an activity as a combination of feature states and actions. Table 2 provides examples to illustrate what a token, place and transition may represent in a Petri Net model. The activity model should be drawn by considering only one instance of an activity. (Many different Petri Nets could be drawn representing the same activity). That instance should start from an initial place where the activity starts and should end at a final place where the activity is completed.

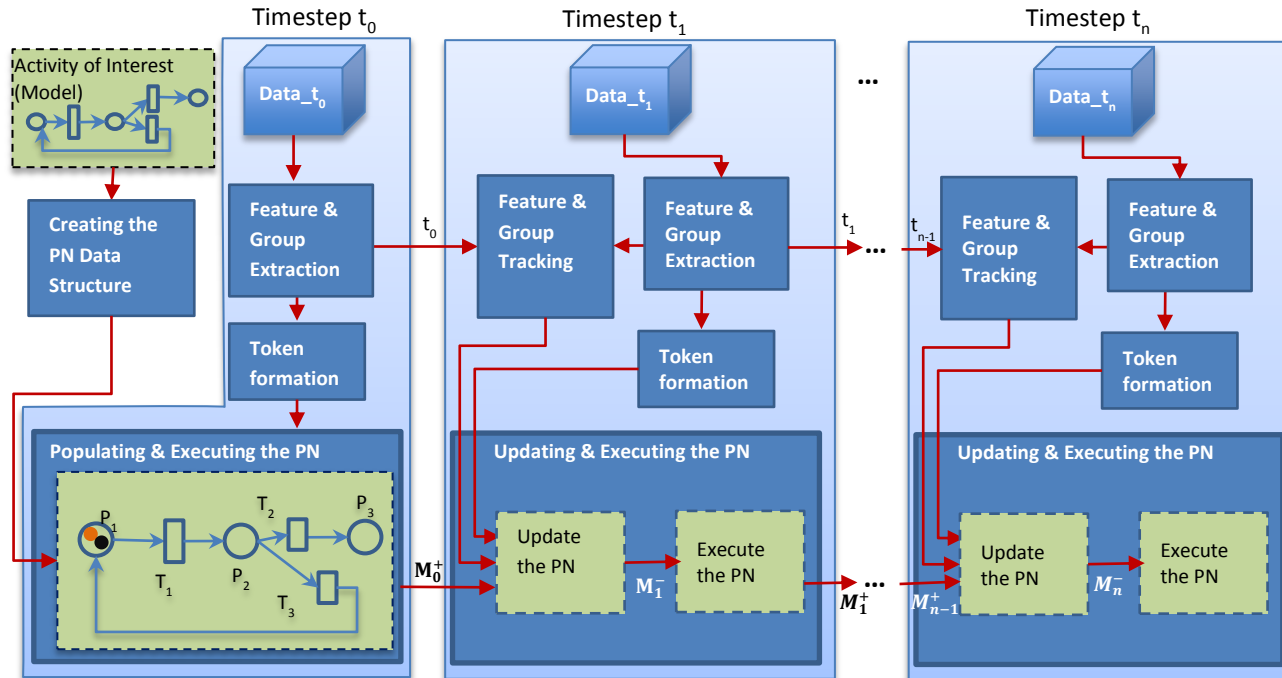


Figure 27: Flow diagram of the proposed activity detection framework. Activity model is given by the scientist and is used to create the Petri Net data structure. Tokens are formed based on the extracted feature (or group) attributes. Once the final marking is obtained, the feature (or group) attributes and the final marking (M_0^+) are passed into the next timestep. In the next timestep, first the features are extracted; their attributes are computed and tokens of the current timestep are formed. Tracking information is combined with M_0^+ to correlate these tokens to the ones in the Petri Net in the update process. This yields the initial marking M_1^- . Once the Petri Net is updated, the execution process yields the final marking M_1^+ . This process recursively continues in each new timestep. The tokens that fall into the final place are the ones that complete the activity.

One important aspect of modelling an activity is that the scientist should consider the flow of tokens from one place to the next, when drawing the model. Since the purpose is detecting multiple events, a token should represent a portion of an instance of the activity in a Petri Net. For example, assume that a place represents “two-people hand-shaking” in computer vision. In that case, a token represents a group of two people who are hand-shaking. Similarly, while a token can represent a feature in one place, in the next place a token can represent a group of features. This is especially useful to simplify the modelling of *formation* type of activities where a feature eventually transforms into a super-structure such as a packet (see [1] and [65]). Since the merge, split, disappear cases are implicitly handled by TTPN (via the sub-net shown in Figure 26c), the scientist does not need to consider these cases in his/her model explicitly for each place. This makes it simpler to define the overall flow.

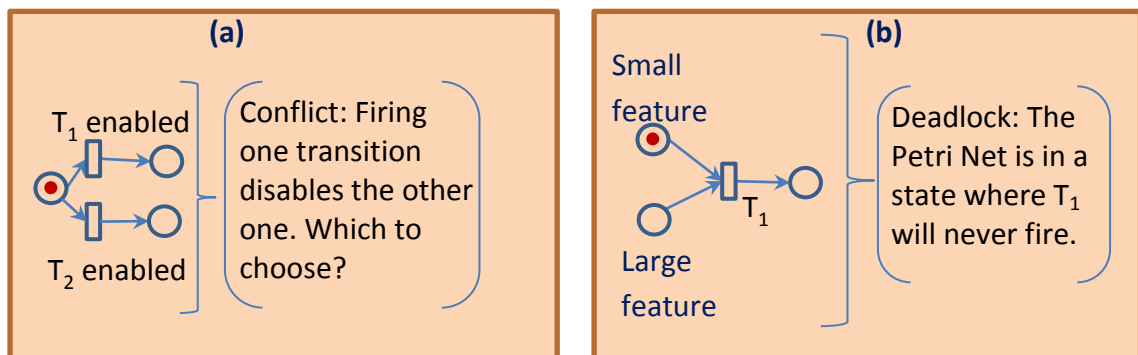


Figure 28: Illustrative examples of (a) conflict and (b) deadlock.

Deadlocks and conflicts: In an activity model, deadlocks and conflicts should be avoided. Figure 28 illustrates deadlock and conflict situations as described in [59] and in [20]. Figure 28a illustrates a conflict case where firing one of the transitions disables the other one. In such situations, our

implementation fires the first transition that is defined in the model. (i.e., the priority is always given to the transition with the lowest transition ID). Figure 28b illustrates a deadlock case where no firing is possible. In this illustration two input places are opposite of each other and it is impossible to be in both of them at once. Therefore, it is impossible to fire the transition. Similar situations should be avoided in a Petri Net model.

Currently, the scientist provides the model along with all the place and transition conditions in a text based *config* file in our existing Petri Net implementation. This is described in detail in Appendix II. We are currently developing a better interface that will help scientist model an activity graphically. Preliminary results can be seen in [80].

CHAPTER 7:

ACTIVITY VISUALIZATION

Activity detection enhances visualization. This chapter presents various visualization techniques that utilize activity detection.

In general, activity detection adds functionality and flexibility to time-varying visualization and allows event based data abstraction. For example, it can identify the timesteps where the activity takes place and the features performing the activity. Moreover, activity detection allows different visualizations highlighting that activity. Different places of a Petri Net can be used to enhance visualization. For instance, the features (tokens) at the intermediate states, i.e., places, can be visualized separately at each timestep. If the scientist is interested in seeing what features from timestep 17 are at place 3 (P_3), those features can be highlighted in an isosurface or volume rendering. Conversely, a scientist can ask at which timesteps features move into P_3 .

Some examples of activity detection visualization are described below:

Graph based activity visualization: In graph based visualization, tokens show the progress of the activity on a given Petri Net graph. An example of graph based activity visualization is shown in Figure 29. The visualized activity is an instance of 15 detected merge-split activities (see Chapter 9.1). The activity model contains three places and three transitions. The visualized instance of the activity starts in timestep 32 and completes in time step 34. In timestep 32, there

are two brown tokens (two brown features) shown below the final place (P_3) indicating that these brown features completed the activity. Two blue tokens and a single red token are shown below the initial place (P_1) indicating that these features are about to merge. The associated features are highlighted in the isosurface visualizations in each time step. The place of a token shows the progress of an activity over time. Therefore, token based visualization is useful to follow the progress of an activity visually.

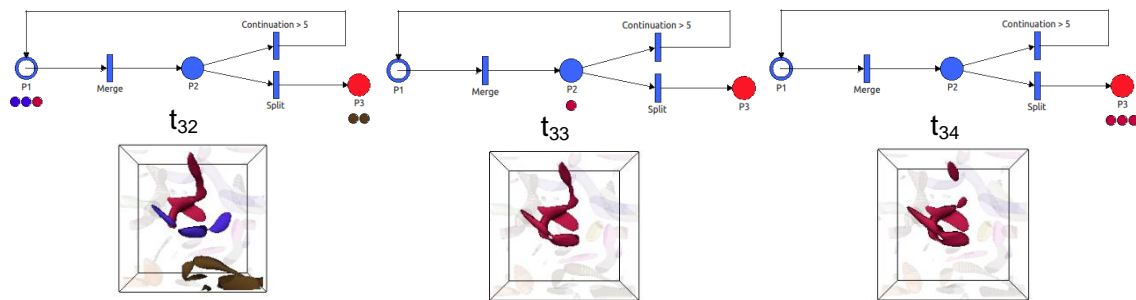


Figure 29: An example of graph based activity visualization. The visualized activity is an instance of the merge-split activity. Tokens are visualized below the places. Associated features are highlighted in isosurface visualization.

As the number of tokens increase in a place, visualizing many tokens in a place may become less informative since the token colors may not be distinguishable visually. In such situations, where the number of tokens is large enough, a histogram (bar-chart) can be attached to each place as an alternative graph based activity visualization as shown in Figure 30.

Activity - Histogram (bar-chart): A histogram (bar-chart) is attached to each place in a Petri Net. The total number of features and how these numbers change over time can be seen in histograms. Example activity histograms are shown in Figure 30. Each histogram visualises the summary of the total number

of features in a place from Figure 29. For example, the histograms under the label P_1 show the total number of tokens in timesteps 32, 33 and 34 individually.

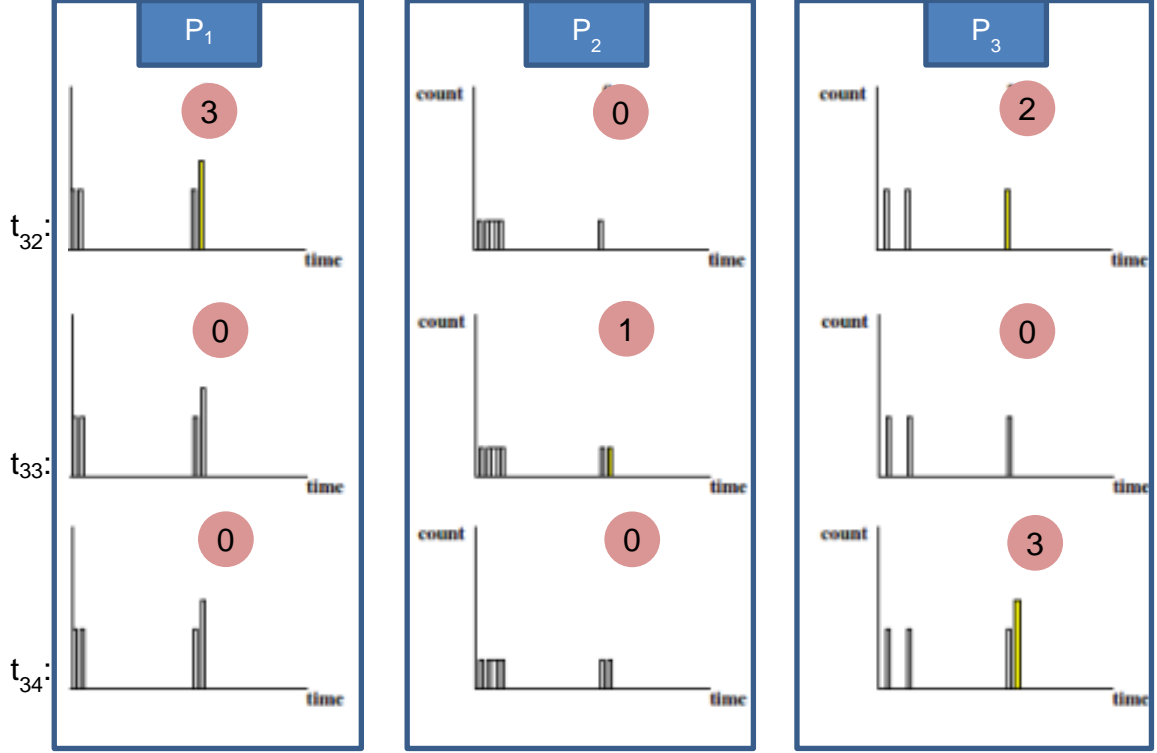


Figure 30: Illustrative example of activity-histogram. An histogram can be attached to each place and the histogram can be updated at each time step as shown. The visualized activity is an instance of the merge-split activity. The activity is previously visualized in Figure 29. The number above each histogram shows the total number of tokens in each place in each time step.

In activity histograms, each bar (at each timestep) can be further segmented, where each segment can summarize the total number of features participating in the same activity. Each of these segments should be assigned a unique color (where the color of a segment may represent either a token or a detected activity). Alternatively, the total number of detected activities can be visualized over time (i.e., the total number of detected activities vs. timesteps) in an histogram. A user can further analyze a specific segment or timestep by selecting

the interesting timestep (or the activity) in the histogram. Thus the selected features can be highlighted in the data and can be visualized.

Activity summary visualization: Activity summary is the visualization of all the detected activities along with the entire data set (or a portion of the data set, if the data set is excessively huge) in a single visualization. Figure 40c is one such visualization of the entire data set. It shows how frequent the activities are and where/when they occur.

In general, activity summary could be visualized in a histogram form, or in a vector form. In vector form, the magnitude can represent the total number of participating features and the angle of the vector can represent the duration of the activity.

Isolated activity visualization: A scientist can also choose to view only one activity from the list of detected activities. Only the features that are currently participating in the user specified activity are visualized. For example, timesteps 70 to 73 in Figure 37 visualize one user specified activity out of the 15 “Merge-Split” activities detected over 100 timesteps. A specific instance of the activity can be selected from the activity histogram. An example of isolated activity visualization can be seen in *Figure 31*. In this visualization, only the selected activity is visualized and only the participating features are highlighted. Alternatively, the participating features can be extracted among all the extracted features and can be visualized individually.

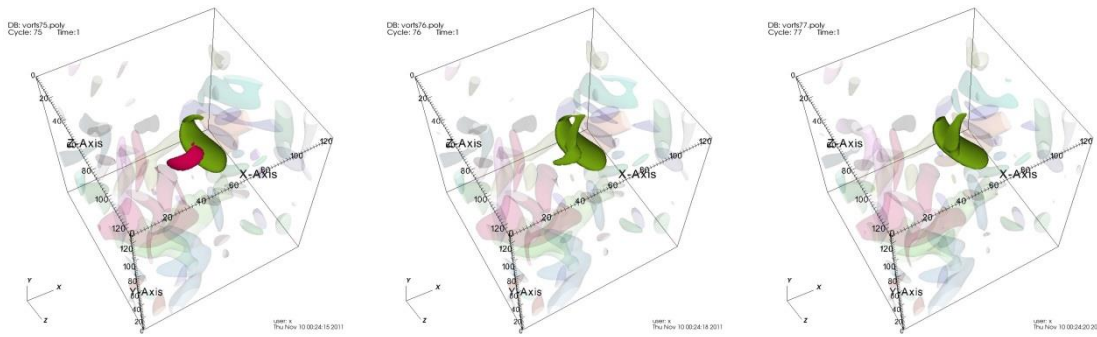


Figure 31: An example of isolated activity visualization. The visualized activity is an instance of the “merge-split” activity (see Chapter 9). There are 15 instances detected and only the selected one is visualized above. This instance happens between the timesteps 75 and 77.

Forecast activity visualization: Forecast activity visualization is the visualization of all the features that “will” complete a specified activity. It visualizes only the features and their evolutions over time performing the modelled activity. In Figure 44a, timestep 8 highlights all the features that are currently performing the packet formation event. This is an example of visualizing features that will form a group in the future. The single feature Feature_A in timestep 8 evolves and eventually forms a group (i.e., the packet Packet_A) in future timesteps.

Activity detection can also help in transfer function design. A time-varying transfer function can be generated by using the activity detection results automatically, i.e., by using the list of the tokens and their activity histories in the final places.

CHAPTER 8:

IMPLEMENTATION

The flow diagram of the proposed activity detection framework is given in Figure 27. The implementation of this framework is formed of two major sub-modules. These are: (1) Feature & Group tracking implementation and (2) Petri Net implementation.

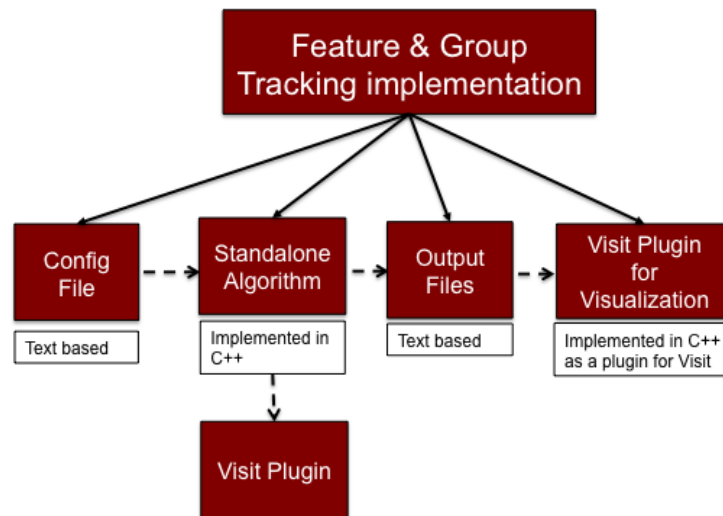


Figure 32: The modules of the feature and group tracking implementation. User inputs the parameters in a text based config file. This file is parsed by the standalone algorithm (which can also run on the Visit platform as a plugin) and accordingly, the data is processed for extracting, tracking the features & their groups and computing the attributes of these extracted features. These attributes are saved in various formats in the text based output files. The visualization module, reads a subset of these generated files to visualize the results.

8.1. Implementation of feature & group tracking

Feature and group tracking implementation reads and processes the data set to extract and track region of interests (i.e, features or their groups), compute the

attributes and visualize their evolution over time. Our implementation is in C++ and works both on Visit (as a plugin) and as a standalone module. Visit plugin is separate from the standalone module. The overview of feature and group tracking implementation is given in *Figure 32*. On the Visit platform, various parameters are given in a GUI environment (provided by Visit). The standalone algorithm uses a text based config file to read the parameters. These parameters include, file (data) names, starting and ending timesteps, variable names to be processed and various thresholds that are used in both feature and group extraction. These parameters passed into the core algorithm which performs feature and group extraction, attribute computation and feature and group tracking based on the given parameters. Once the core algorithm, the “standalone algorithm”, completes running, the features and their groups are visualized on Visit via a plugin which uses the output files of the standalone algorithm as input.

The standalone algorithm saves the computed information in multiple files based on the information type. All these files are saved under a user specified folder (For the Visit plugin, this is a folder with the name *GENERATED_TRACK_FILES* which is created under the data folder). Based on the information type, the computed information is saved in **.poly*, **.trak*, **.attr*, **.uocd*, **.trakTable*, *t.groupTrakTable*, and colormap files. Detailed information for the format of these output files can be found in Appendix II.

The core output files (based on their extensions) are the **.poly*, **.trak*, **.group* files where *** represents the actual data file name with the individual

timestep index. For example, for a data set including “*data1.vtk*, *data2.vtk*, *data3.vtk*” we would have “*data1.poly*, *data2.poly*, *data3.poly*, *data1.trak*, *data2.trak*, *data3.trak*, *data1.group*, *data2.group*, *data3.group*” etc. Each *.poly* (for example *data1.poly*) file contains the surface information of all the extracted features. The surface information is saved in mesh format as in the earlier versions of feature tracking implementations. Besides the surface information, if the individual node values are also needed, then the generated **.uocd* files can be used since they also include node values within each feature. In addition to these files, there is also a single *.trakTable* file which saves the entire tracking information of the features in all timesteps. Similarly, the file *t.groupTrakTable* saves the group tracking information for all the groups in all the timesteps. Both feature and group tracking files use “-1” as a key term to separate the previous timestep object IDs from the current timestep object IDs. The **.trak* files include various attributes of each feature. Each line in a **.trak* file contains various attributes of a feature. These attributes include the centroid location, feature volume, mass, moments, min and max values and the coordinates of these min and max values. In addition to these attributes, the bounding box coordinates summarized with two point locations (located on the lower left and the upper right corners of the bounding box) and group IDs of each feature are also included in these files. The order and the number of these computed attributes can be adjusted based on the domain. **.attr* files present these computed attributes in a more human readable form. **.group* files include group information. Each line in a file represents one group (where group ID is the line number) and includes the

member feature IDs from the same timestep. Therefore the number of objects change from one line to the next in **.group* files.

The feature extraction is done by a region growing algorithm which is re-implemented by using VTK libraries. The feature tracking algorithm fundamentally relies on the implementation in [71]. The key point of the new region growing implementation is that the most computational gain is obtained by the removal of process of searching for the local maximum points. These local points were used as being the seed points in the original region growing algorithm. To save computation and time, our algorithm searches along each node only once in the data and when it finds a nodal value that is greater (or smaller) than the given threshold, it starts growing around that node to extract the feature. Since a feature is a set of connected components, this approach yields the same results when compared to the earlier versions of the region growing algorithm in [71] while gaining a huge improvement on the computational side. In the group tracking implementation, user defined similarity functions define the groups. This similarity function is used within a clustering algorithm to determine the groups. The determined groups are tracked based on the feature overlap criteria.

The computed attributes and tracking information are used in the Petri Net implementation for activity detection.

8.2. Implementation of TTPN

The TTPN implementation takes the Petri Net model (given by the scientist) and the computed meta-data as input and creates a text based output file that

lists the detected activities. This file is used for visualization and for further data analysis. Figure 33 shows various steps of our Petri Net (PN) implementation. The first step is activity modelling. The activity model (the Petri Net model) is saved in a config file. The second step is activity detection. The config file is passed into the PN algorithm. The PN algorithm processes the meta-data over the time to detect activities and creates a text based output file. The final step is activity visualization. The activity visualization step uses the list of the detected activities (the output file). Current activity visualization implementation runs on Visit platform as a plugin and visualizes the features in the timesteps specified in the output file.

In our Petri Net implementation, which is implemented in C++, the Petri Net data structure is formed according to the model given by the scientist. In a given Petri Net model, the tokens are the only variables/classes that change over time. Each token also has a *token-history*. A token history is a list that adds the triple tuple (t_j, P_i, ObjID) to a token's token history at each iteration where the t_j is the j^{th} timestep and ObjID is the object (token) ID.

In the merge case, all the merging object IDs form their individual triple tuples in a token-history. In our TTPN implementation we use logical or mathematical expressions formed of object attributes to describe a feature's state or action. A place condition is run at each timestep to determine whether a token still remains in that place. Tokens that change their states are put into a vector for a further evaluation to check if they changed their places via the firing process. A transition condition is used to determine whether that transition can be enabled

for a given token. If a token satisfies the transition condition, then a second step checks whether the same token exists in all the incoming places. Furthermore, a third step checks whether the object satisfies “at least” one of the output places’ conditions. After passing the third step, the transition is enabled and ready to fire. Firing a transition for a token removes the token from all the incoming places’ lists, and inserts it into the output places for which the token satisfies the place conditions.

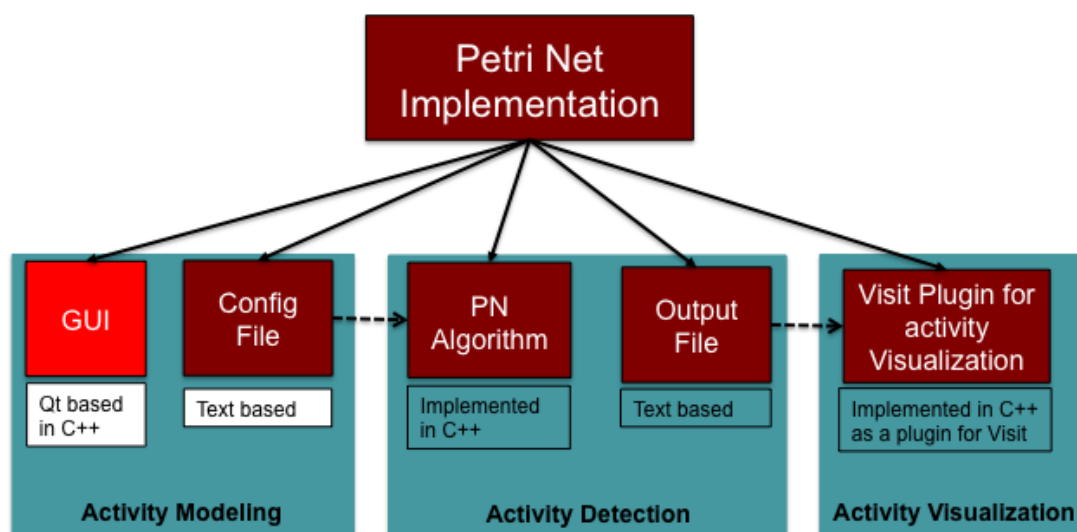


Figure 33: The modules included in the Petri Net implementation are shown. These modules include activity modeling, activity detection and activity visualization. Activity modeling module includes the user interface that saves the activity model in a format that the PN algorithm can read. The format is given in a text based config file. The activity detection module reads the config file and creates the data structure accordingly. Then the PN algorithm is run on the meta-data. The detected activities are saved in a text based output file. This file is passed into the activity visualization module in which first the output file is parsed to obtain the participating feature IDs and their associated timesteps; then these parsed feature IDs are highlighted in the appropriate visualization technique (see Chapter 7 for a list of activity visualization techniques).

In each domain or in each application, different feature attributes can be computed and saved in different orders. Moreover, a different combination of the available feature attributes can be used as a condition for each place or for each transition. To cover such variability and flexibility in action and state definitions, in our implementation we use *Petri Net variables*. A Petri Net variable is either a specific feature attribute or a default action from a library (such as merge, split, continuation or new born) and can belong to either the current timestep or the previous timestep.

In our implementation, Petri Net variables take one of the following forms: “ $t_c A_\#$ ”, “ $t_p A_\#$ ”, “ $t_c D_\#$ ” or “ $t_p D_\#$ ” where the first two characters, t_c and t_p , stand for the current timestep and previous timestep respectively. $A_\#$ is an integer number and represents the index (column) number of an attribute from a list of attributes for a given feature and $D_\#$ represents the index number of the predefined action from a library. For example, “ $t_c D_4$ ” means the fourth action from the library (which is the split action in our implementation) in the current time, and “ $t_c A_3$ ” means the third attribute value of a feature in the current timestep.

Let us consider the transition: “*Volume increase is more than %40 of the previous volume value*”. This can be expressed as a difference of the volume values of the current and previous timesteps. Assuming the third value in the attributes file represents the volume, we can construct the related transition condition as “ $(t_c A_3 - t_p A_3) > (0.4 * t_p A_3)$ ”. This Boolean expression decides whether the condition is satisfied and serves as an *action* detector. Similarly, the place conditions can define the feature states. Our token-tracking Petri Net

implementation allows the use of built in functions for constructing similar condition expressions.

Activity visualization uses the token history. In our implementation, the token history is captured in an output file. In the output file, each line represents one detected activity. Each of these lines formed of a sequence of triplets. The sequence represents the activity model. Each triplet includes an object ID, its current place ID, and the current timestep. The difference between the first and last triplets' timesteps yields the activity duration.

Figure 34 illustrates the data structure of a Petri Net. The Petri Net data structure includes various types of objects. These include transitions, places, arcs and tokens. Since the number of transitions, places and arcs change from one PN model to another, linked lists (and vectors in C++) are the data structures we use in our implementation. Each arc is defined as triplet including the arc-weight, and the IDs of its starting and ending nodes that are formed of one place and one transition. While all the arc weights have the value of one in TTPN models, we still keep the arc-weight variable in each arc for compatibility with other Petri Nets. Each transition has its own ID, condition, a list of its incoming arcs and a list of its outgoing arcs. Similarly, each place has its own ID, condition, a list of its incoming arcs and a list of its outgoing arcs. In addition to these members, places also have a list of its tokens and a flag indicating whether the place is an initial place or a final place. Each token has its own ID, its history, a set of attributes that are used to define the Petri Net variables in conditions, and

a flag indicating whether a token is still active in the Petri Net or not (for disappearing tokens or searching purposes).

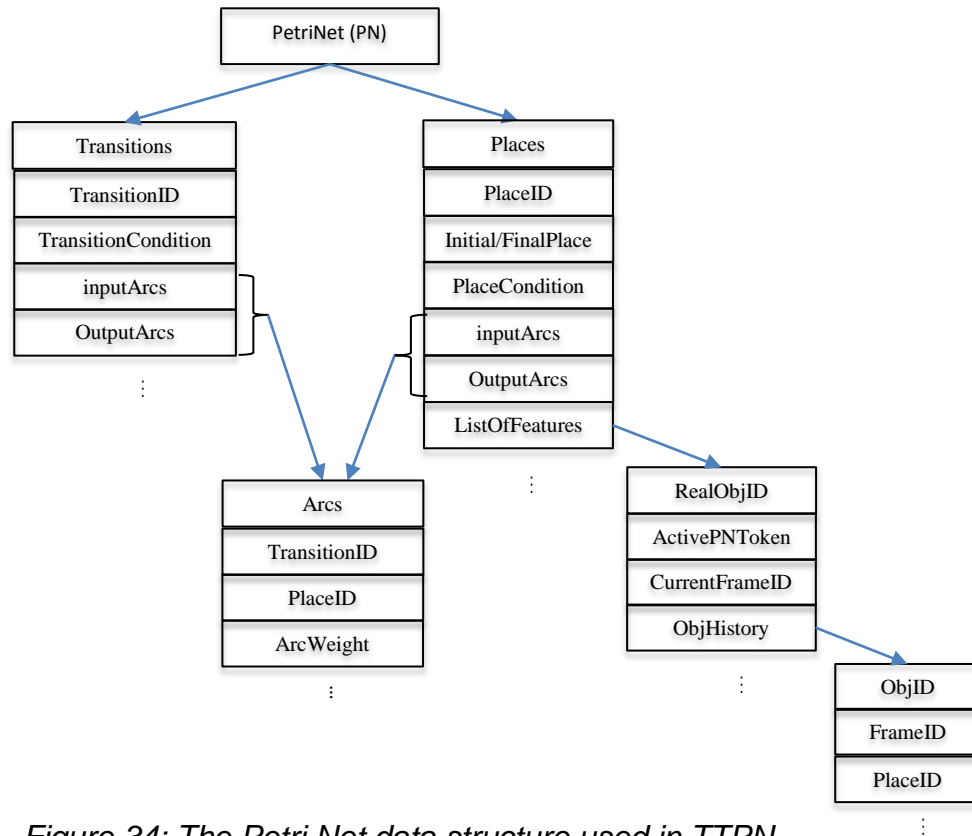


Figure 34: The Petri Net data structure used in TTPN.

CHAPTER 9:

EXPERIMENTAL RESULTS FOR ACTIVITY DETECTION

Activity detection is applied to three different case studies. Each of these case studies uses a different data set.

The data sets used in this dissertation are described in Appendix III.

9.1. Merge-split activity detection in turbulent vortex structures

To test the TTPN, we first apply it on the pseudo-spectral vortex simulation data (see Appendix III-A3.1). The activity of interest is a “*Merge-Split activity*” where a single vortex merges with another vortex and then splits again within k_0 time frames. This activity is similar to the ones found in [12] and [24]. A Petri Net for this activity is shown in Figure 35. The variable k_0 represents a duration set by the user. In this Petri Net model, if a feature performs the continuation event more than k_0 consecutive time steps, then the feature (token) goes back to P_1 .

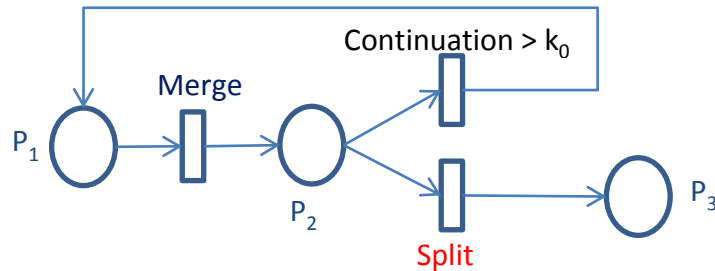


Figure 35: The Petri Net model of the merge-split activity. In this model, first multiple features merge to form a single feature and then within k_0 timesteps, the merged feature splits into multiple features again.

Notice that the definitions of the places are not important. Therefore all the place conditions set to logical *true*. The transitions characterize the entire activity. Each transition is an event that is detected by processing the tracking history. For testing, k_0 was set to 5 in our applications. However, we include how the total number of detected activities changes with respect to the k_0 value in Figure 36. In the figure, the k_0 value changes from 1 to 10 along the x-axis and the total number of detected activities at each k_0 value is shown as a blue diamond. As shown in the figure, the total number of detected activities increases as the value increases except the $k_0=5$ value, at which the total number of detected activities remain the same when compared to the value at $k_0=4$.

The simulation data resolution is 128^3 and the data contains 100 timesteps. The features and their attributes along with their tracking history were computed by the feature tracking algorithm at the threshold value 5. The merge, split and continuation events of each feature is computed from the correspondence list. The correspondence list is computed by the feature tracking algorithm. Running TTPN on this meta-data found 15 completed “Merge-Split” activities in the 100 timesteps. Figure 37 visualizes three sample activities. As it is shown between the timesteps 64 and 67, our proposed technique is not limited to detect only one activity at a time. These two detected activities overlap in time. On the figures, each feature has a distinct color, except that splitting features have the same color. Only the features that are currently performing a detected activity are highlighted and all the other features are visualized transparently. Figure 38 visualizes another two instances from the detected 15 activities. The first one is

between the timesteps 1 and 3. The second instance takes place between timesteps 3 and 8.

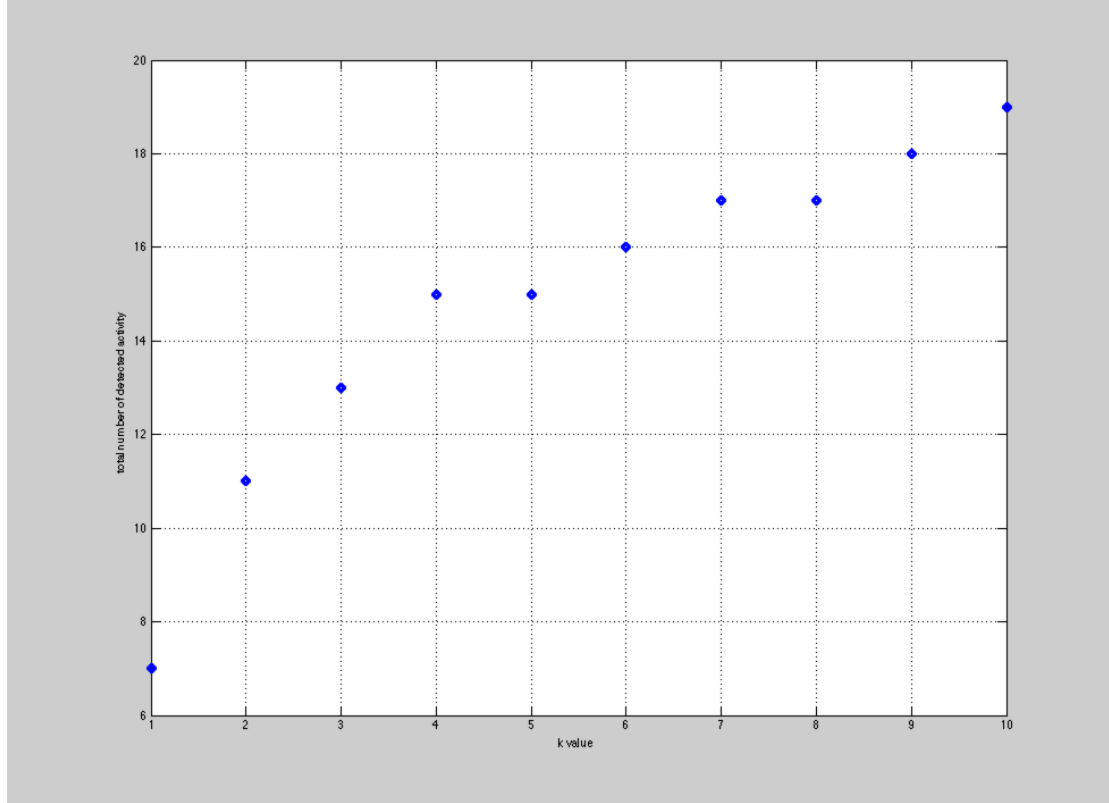


Figure 36: The total number of detected activities changes, as the value of k_0 changes. The value of k_0 is changed from 1 to 10 and for each k_0 value, the total number of detected activities is plotted with a blue diamond on the plot.

While all the visualizations in this thesis are obtained at fixed k_0 value ($k_0=5$) for the Merge-Split activity, we also include another combination of merge and split events in Figure 39. Figure 39a shows the Petri Net model describing the “merge-split-continue-split” case. TTPN detected 8 instances of the modeled *merge-split-continue-split* activity in 100 timesteps and these instances are visualized as histograms in Figure 39b.

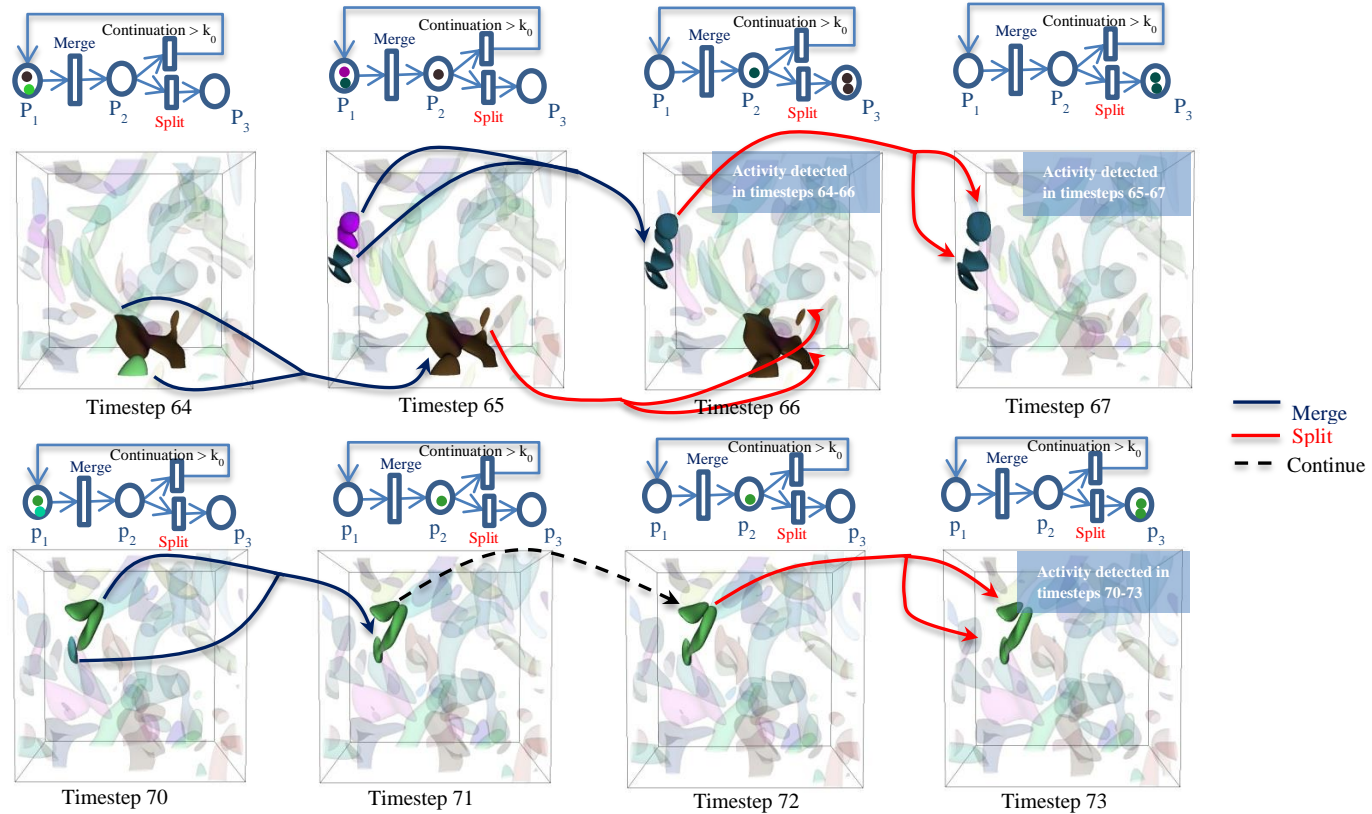


Figure 37: Three detected instances of the “merge-split” activity are visualized. The activity is modeled as a Petri Net shown above, and is defined as a feature merging and splitting within $k_0=5$ timesteps. The activity detection process found 15 “Merge-Split” activities over 100 timesteps. Three activities out of those 15 are visualized above. The colored dots in the Petri Net show the locations of the participating features for each timestep in the associated feature colors. All other vortices that do not participate in an active merge-split activity are shown transparent in the visualization. Two of the found activities are shown in timesteps 64-67. The associated time-varying transfer function is automatically generated for the visualization. Petri Nets encapsulate the components that define an activity and help in abstracting time. For example, another “Merge-Split” activity which occurs over 4 timesteps instead of 3 is shown timesteps 70-73.

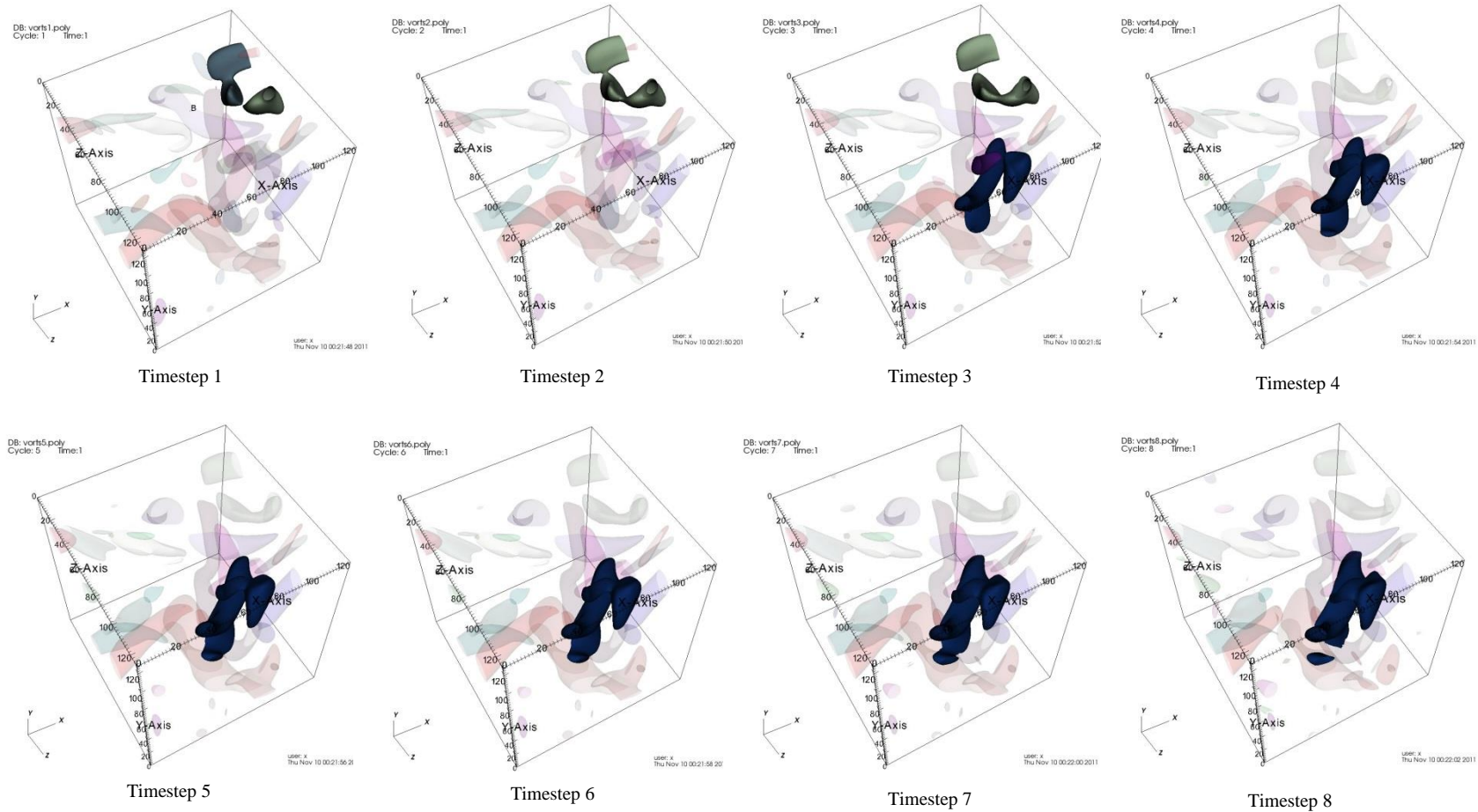


Figure 38: Two other detected instances of the "merge-split" activity are visualized. The first detected activity is shown between the timesteps 1 and 3. The second detected activity starts at the timestep 3 and completes in timestep 8.

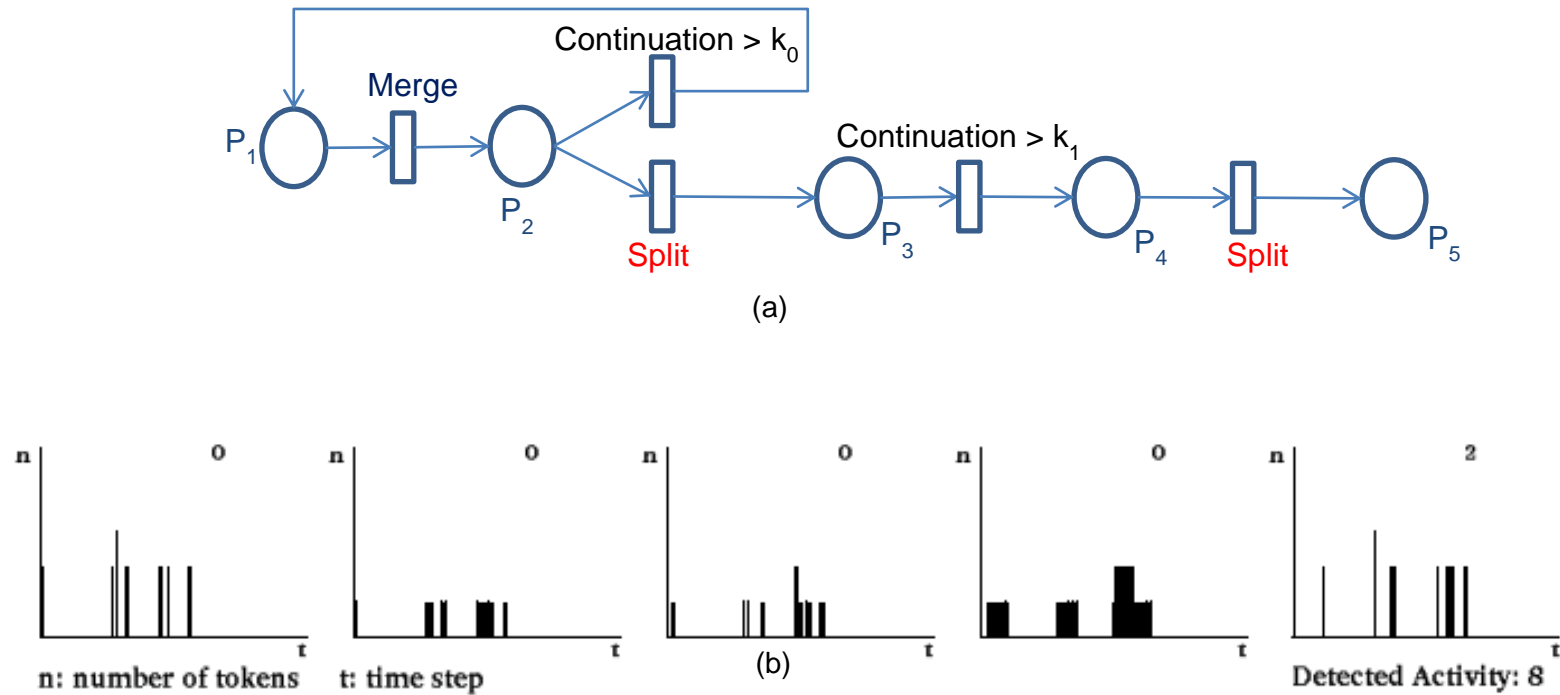


Figure 39: Another combination of merge, split and continue events is modeled. (a) shows the Petri Net model for a longer activity. In this activity, we seek for a subset of features that have completed the “merge-split” activity. In addition to performing previously defined “merge-split” activity, features also need to continue for at least k_1 times and then split again. (b) the results of the modeled Petri Net are summarized in histograms. Each place is assigned a histogram. After running the Petri Net on 100 time steps, TTPN detects 8 instances of the modeled activity. These results are obtained when $k_0=5$ and $k_1=1$.

9.2. Detecting anomalous bending in acoustic plume scans

For the next application, we have used the meta-data computed from the acoustic plume scans. Detailed description of the data can be found in Appendix III-A3.2. The meta-data is obtained from [66]. We have applied a Petri Net analysis to determine if the bending patterns observed in the plumes are consistent with a semi-diurnal tidal cycle (the datasets have multiple plumes, in this example, we focus on only one of them). Ordinarily, changes in direction of tidal flow imply a stagnant period in between directions during which the plume would be vertical. Ocean currents are not expected to shift the plume from left to right or vice versa by skipping a state between two timesteps along the x-z plane. By defining each direction of plume bending as an object state (Figure 40a) we use a Petri Net to model the normal process and detect the anomalies along the x-z plane (Figure 40b). We observe three instances of anomalous-bending at times steps 9, 10 and 12 (circled in red in Figure 41a) within the first plume data set. Our TTPN algorithm provides results that match the observed results.

However, when applied to a larger data set, the results of the current Petri Net model were too large. This let the scientist to refine the model since the original model did not consider the 3D nature of ocean currents and plume responses. The redefined new states with new conditions are shown in Figure 42a and they consider the angles in 3D between z axis and x-y plane. The scientist observed that the plume can move left or right within 45° (along the x-y

plane) and up or down (along the z axis) from one state to the next. Any angle difference that is greater than 45° is considered as an anomaly. The new TTPN (Figure 42b) has detected 131 timesteps as anomalies out of the 479 available timesteps. The available 479 timesteps are plotted along the x axis (where the unit is in days) in Figure 42c. For each day, the magnitude of Plume A bending and its major direction is represented using an arrow. The anomaly events are highlighted with a pink star. While this is still large, it is much more in tune with the data. Further discussion on plume bending can be found in [8].

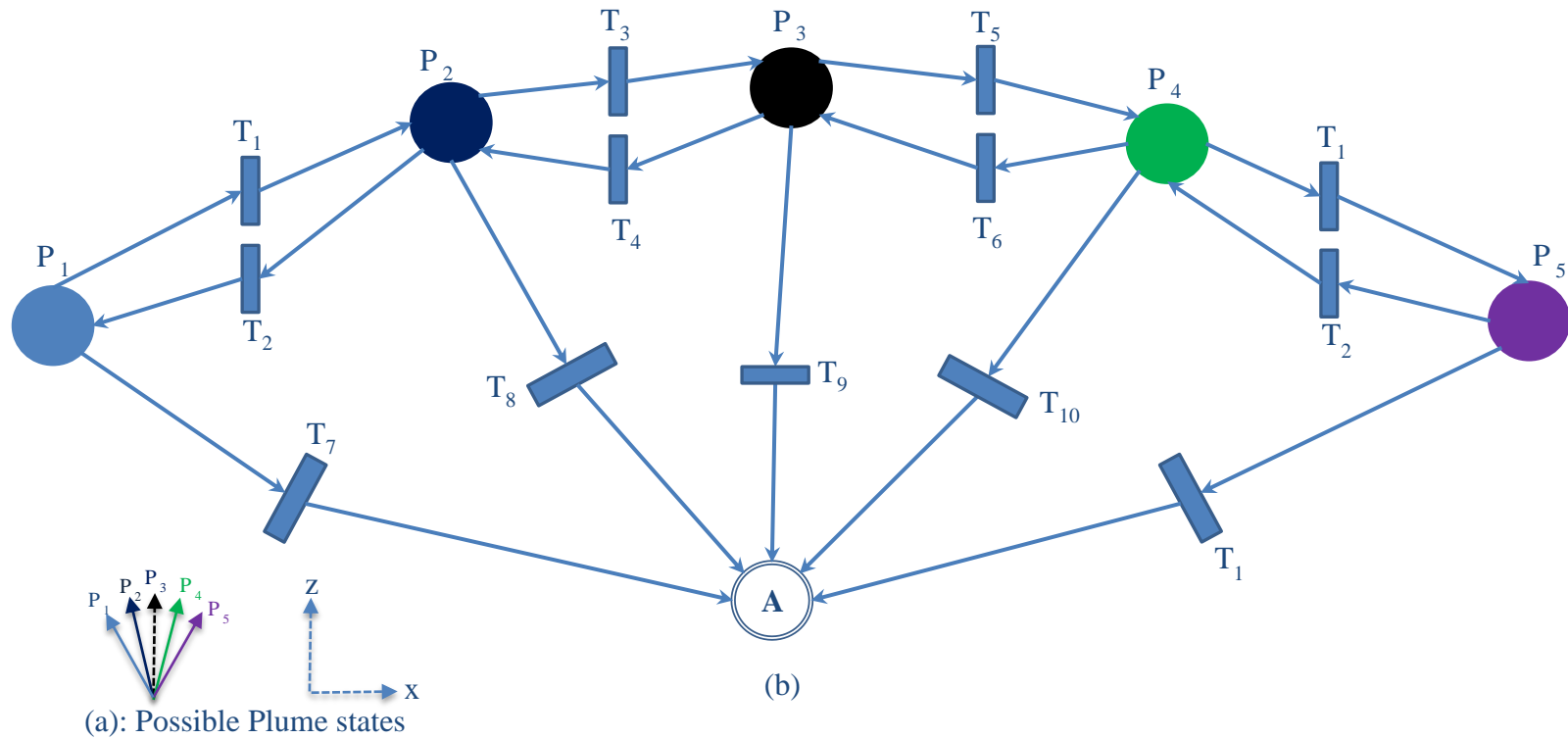


Figure 40: A Petri Net model for “Anomalous Plume bending” is shown. (a) The plume behavior is categorized in 5 states (P_1 , P_2 , P_3 , P_4 and P_5) based on its angle (the angle between the center line of the plume and the z axis). These five states are illustrated in two dimensional plane. (b) Based on the defined five states in (a), the Petri Net model is drawn. The place that is labeled “A” represents the anomalous bending and therefore when the token (i.e., the timesteps in this case since there is only one plume) falls into this place, an anomalous bending is detected. This Petri Net is applied on 15 timesteps.



Figure 41: “Anomalous Plume bending” detection in a time-varying 3D plume data set is visualized. This data set is formed of 15 timesteps (i.e., 3-dimensional scans of the plumes are obtained at 15 different times). This data set contains one of three available plumes. (a) In each timestep, first the plume is segmented by using the feature tracking algorithm in the data and the isosurface of the extracted plume is visualized. The relative orientation of the plume to the normal is shown on the right of each timestep. The anomalies that do not fit the defined periodic movement between the plume bending states are circled in red. (b) In this data set, there are 5 states are defined and these states are shown in x-z plane, i.e., the states are defined in 2D for these 15 timesteps. There are two states on each side of the normal.

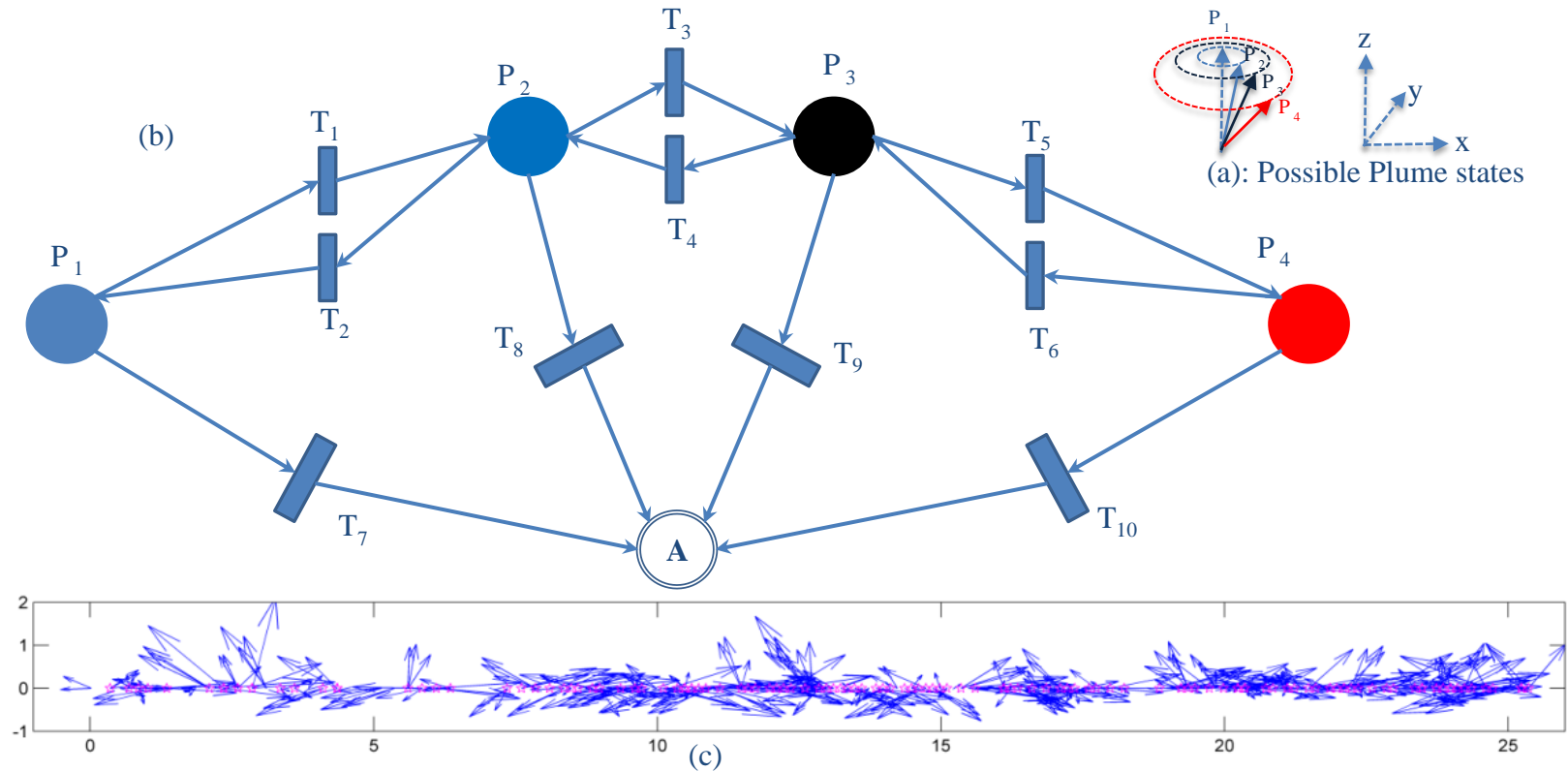


Figure 42: The Petri Net model for “Anomalous Plume bending” is shown. (a) The plume behavior is categorized in 4 states (P_1 , P_2 , P_3 and P_4) based on its angle (the angle between the center line of the plume and the z axis). These four states are illustrated in three dimensions. (b) Based on the defined four states in Figure 40a, the Petri Net model is drawn. The place that is labeled “A” represents the anomalous bending and therefore when the token (i.e., the timesteps in this case since there is only one plume) falls into this place, an anomalous bending is detected. This Petri Net is applied on 479 timesteps. (c) The plume’s direction (and magnitude) is shown over 26 days (total 497 timesteps). The pink stars are the anomaly timesteps in which the anomalous plume bending is detected by the Petri Net.

9.3. Packet formation in wall bounded turbulence flow simulations

In Direct Numerical Simulations (DNS) of wall bounded turbulence flow, scientists have been interested in searching for the existence of groups of coherent but unconnected features, their formation, dynamic evolution and number of these groups [65], [53]. An illustration of such a group is shown in Figure 44c [1]. The yellow hairpin vortices (features) move coherently inducing a secondary (blue) fluid mass of low momentum. These coherent structures are called packets. (This is analogous to groups of humans walking coherently in a crowd or to a school of fish). In general, the hairpins are not connected within a packet. Each packet includes varying number of hairpins where these hairpins are aligned at a downstream-leaning angle (γ) and the distance between the hairpins should not exceed a predefined physically meaningful value. Some of these packets lead to the formation of younger packets over time. Moreover, among all these packets, some act coherently forming super structures inducing meandering regions of low momentum. (an illustration of such super structures is given in Figure 44c) The activity we are interested in is the “packet formation” event which is characterized by a single hairpin evolving into a packet formed of multiple features over time [65].

The initial simulation data (shown in Figure 17 and in Figure 44a) has 46 timesteps with the resolution 384x256x69. Figure 44b shows the Petri Net model

for packet formation. In Figure 44b, P_1 and P_2 represent a packet formed of a single feature, P_3 and P_4 represent a packet formed of multiple features in Figure 44b. The activity (packet formation) starts at P_1 (initial place) and ends at P_4 (final place). Notice that the transition “A group of hairpins moving together” can be replaced with another Petri Net to detect and identify groups. Group dynamics needs to be computed as a part of the tracking algorithm. See Chapter 4 for the details of group tracking and group dynamics.

Quantification and visualization of the packets in time-varying data require us to track the history of the packets. Feature extraction is performed at the threshold 0.1×10^{-3} via a region growing algorithm and the objects with the volume lower than 25 are filtered. The average number of extracted features is 308 and the average number of found groups is 163 in 46 time frames. Figure 44d demonstrate the number of found feature and packet numbers in each time frame.

The PN model yielded 288 packet formation activities over 46 timesteps. Figure 44a visualizes the portion of the activities that take places in the timesteps between 8 and 13. It is apparent that the single Feature_A (circled in purple) transforms into the Packet_A in the following timesteps. All other packets that are not currently performing the modeled activity are transparent (Forecast activity visualization).

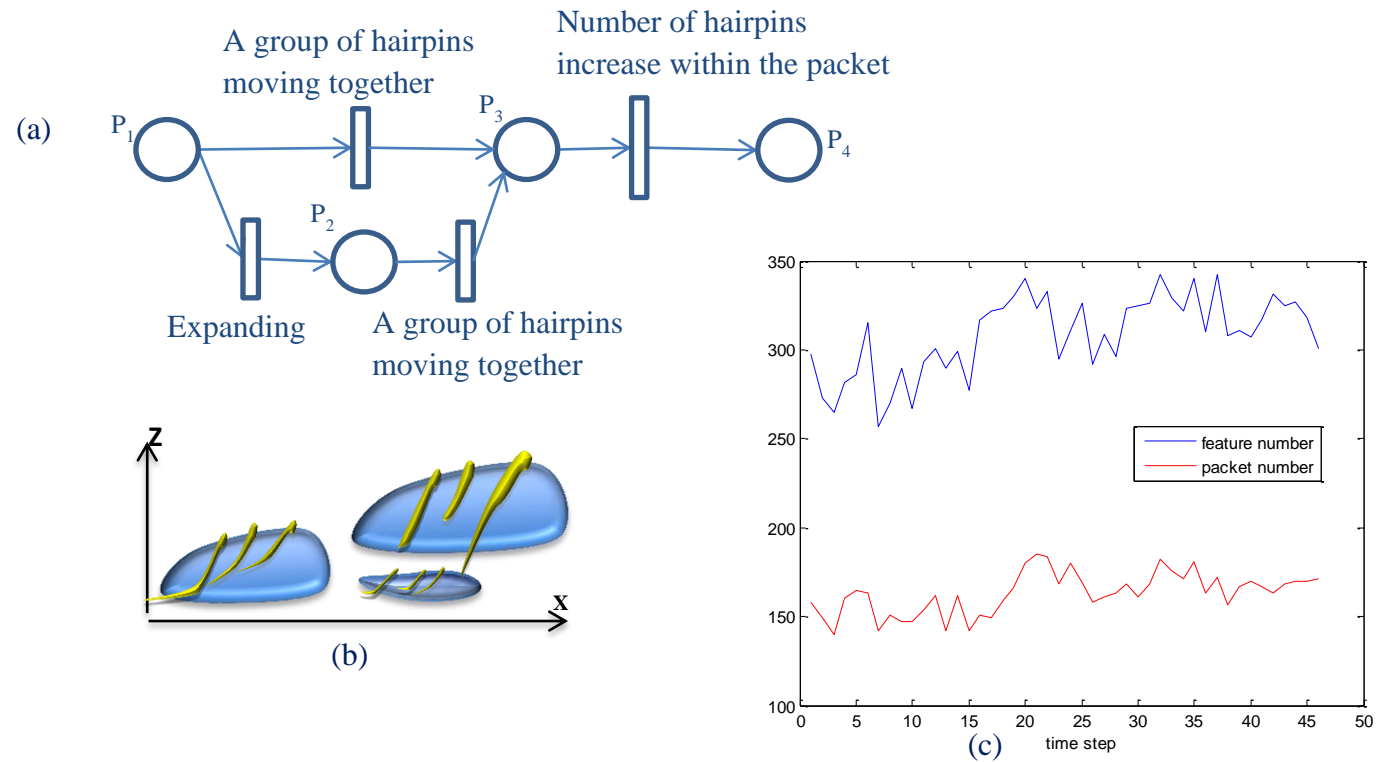


Figure 43: Various information of the wall bounded turbulence DNS, (a) A PN model for the packet formation event where p_1 and p_2 represents single hairpin vortex, p_3 and p_4 represents packets including multiple hairpin vortices, (b) an illustration of a packet (a group of hairpin vortices) and a super structure formed of packets, (c) The number of found packets and features (hairpin vortices) in each timestep.

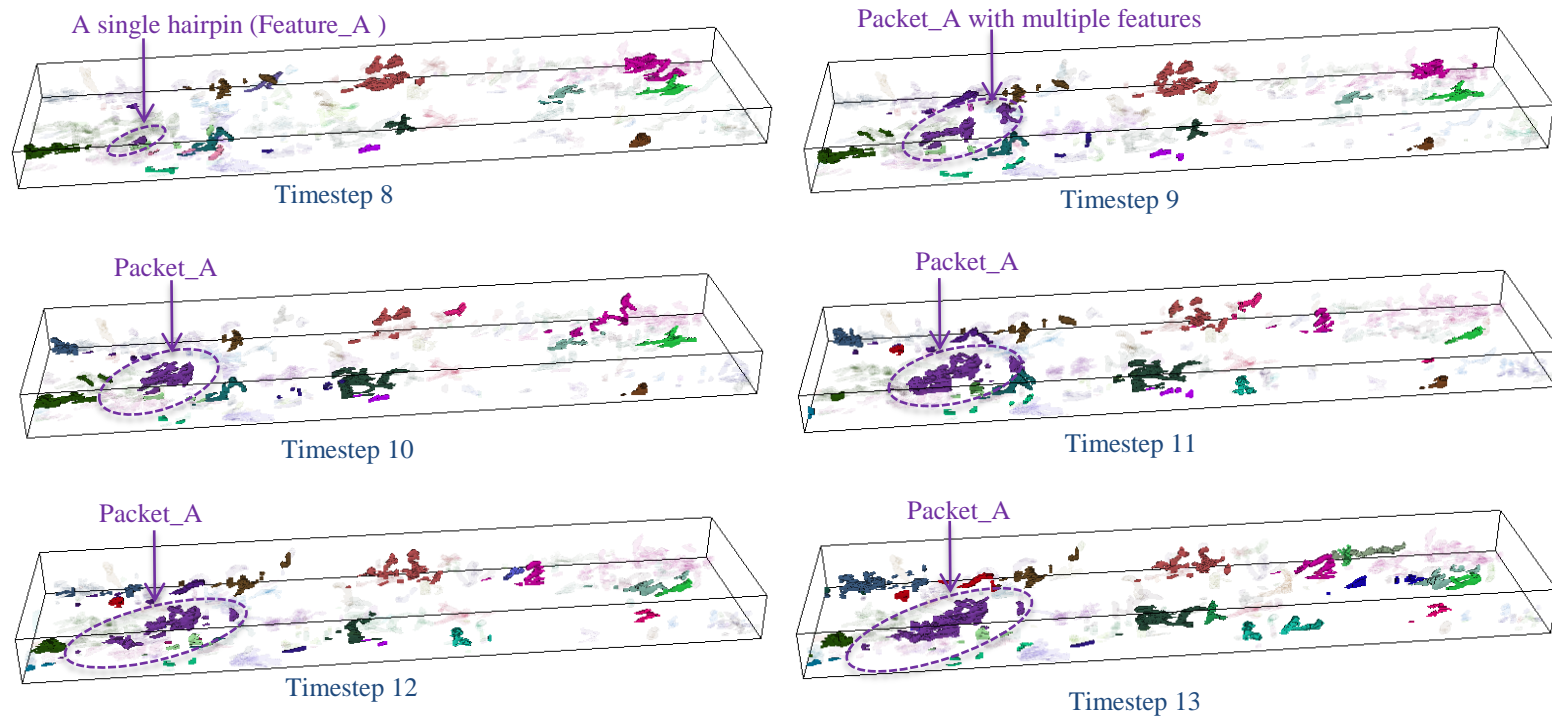


Figure 44: Sample detected packet formations are visualized in wall bounded turbulence DNS. An example formation of a packet (Packet_A) is circled in purple. In timestep 8 the purple feature is a small feature while in timestep 13 it looks like a bullet shaped packet (A packet is illustrated in the previous figure).

CHAPTER 10:

DISCUSSION AND CONCLUSION

In this dissertation, we show that there are activities in 3D scientific simulations and these activities can be modelled and detected by employing the available activity detection techniques from computer vision applications. An activity can be modelled as a time sequence where each node in the sequence represents a state of an object. In a high level description, activity detection is extracting the meaning from the data.

In this dissertation, the use of a graph based technique, Petri Nets, is proposed first to model an activity of interest and then to detect the instances of the modelled activity over time for 3D time-varying scientific data sets. The proposed technique relies on the object attributes. These attributes are computed by first extracting the objects and then tracking them over the time by the available feature and group tracking algorithms. As a part of this dissertation, a group extraction and tracking algorithm is presented to determine the groups and to track them over the time.

Petri Nets operate on tokens. In scientific visualization, the tokens correspond to the features including feature attributes. In scientific simulations, feature attributes change over the time, therefore the tokens also change over the time. In order to include such time variance, an enhanced Petri Net: token-tracking Petri Nets is proposed in this dissertation. Token-tracking Petri Nets can handle the variance in an object's state or the change in its attributes as going

from one timestep to the next. Token-tracking Petri Nets utilize the available tracking information to update the tokens in each place. In this formalism, each token gains time-varying capabilities.

Activity detection with Petri Nets helps scientists with hypothesis validation in scientific simulations. A scientist can first formulate an idea or an hypothesis of how features interact or how they evolve and then search for that activity amongst thousands of time steps. Through an iterative process, the hypothesis can be refined by interpreting the visualized results.

Future work may include integrating the activity detection results within the visualization step in a more efficient way. There are various ways of utilizing the results of activity detection in visualization (these are partially discussed in Chapter 7). Effective routines can help visualizing large data sets on standard computers within a reasonable computing time.

Group tracking is an important part of the proposed activity detection framework. Increasing the grouping accuracy and the tracking accuracy helps detecting the activities more accurately. The visualization of the group tracking results can also be enhanced. The current version relies on retaining the same color information of the dominant parent. These may results in visually misleading results in some cases, where all the objects (features) initially originate from the same group or the same feature. In such situations all those child features will retain the same color even if they are in different groups. Various different visualization approaches can be applied to avoid such confusion.

In this dissertation, all the applications were based on physically observable coherent features. However, notice that token-tracking Petri Nets are not limited to the detection of the activities of only coherent features. The proposed activity detection framework can also be applied to detect the activities of specific nodes or quantities in both Lagrangian and Eulerian simulations. For example, activities such as “the minimum pressure remains constant for 5 time steps” can still be modelled and detected by token-tracking Petri Nets. In this case, the segmentation and tracking steps of the framework would become trivial since either each node, a quantity or the entire domain would become a token in TTPN.

One of the main challenges in scientific data analysis is creating the training set or ground truth for the use of available machine learning or data mining techniques in scientific simulations. Petri Nets can also create the necessary training data set from the semantic descriptions for further analysis with other data mining techniques. A semantic based approach (such as Petri Nets) allows exploratory knowledge discovery besides detecting certain events in timevarying data sets. Once a good set of (representative) training data is created, future work may focus on including machine learning techniques for object recognition, object state detection or for action detection within the proposed framework. Classification algorithms could be useful for recognizing different object types or different object actions.

The use of activity detection is demonstrated in three sample data sets. The different case studies demonstrate that while the domains and actors are

different, the concept of activity detection can be applied to all. The proposed activity detection framework went over all the time steps and pulled out the relevant features and time steps effectively into more manageable chunks. Experimental results showed promising results to use Petri Nets for “knowledge-assisted visualization”. Therefore graph based techniques work in conjunction with the available visualization techniques and they remain as intuitive and accessible solutions to the both scientists and the visualization community.

TTPNs support parallel and distributed systems and thus parallel processing. Due to their graph based structure, TTPN remain as a scalable choice for modelling activities performed by multiple features. Future work may include investigating the optimal parallel computational approaches for TTPNs.

ACKNOWLEDGEMENTS,

PUBLICATIONS AND PRESENTATIONS

I acknowledge that the data used in this dissertation is obtained from our collaborators including Prof. Pino Martin (for the wall bounded turbulent simulation data), Dr. Karen Bemis (for the under-water plume data) and my advisor Prof. Deborah Silver (for the pseudo-spectral simulation data).

The following is the list of publications and presentations including a portion of the material presented in this dissertation:

- 1) **S. Ozer**, D. Silver, K. Bemis, P. Martin, "Activity Detection in Scientific Visualization", Visualization and Computer Graphics, IEEE Transactions on, 2013: ***This publication is a summary of the various chapters included in this dissertation.***
- 2) L. Liu, **S. Ozer**, K. Bemis, J. Takle, D. Silver, "An Interactive Method for Activity Detection Visualization", Large Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on, 2013 (**Poster**).
- 3) **S. Ozer**, J. Wei, D. Silver, K.-L. Ma, P. Martin, "Group Dynamics in Scientific Visualization", Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on, 2012: ***This publication is a summary of Chapter 4 fundamentally. Therefore, this publication includes the content from Chapter 4 mainly.***

- 4) K. G. Bemis, **S. Ozer**, G. Xu, P. A. Rona, D. Silver, “Event Detection for Hydrothermal Plumes: A case study at Grotto Vent”, **Abstract** OS52A-05 presented at 2012 Fall Meeting, AGU, 3-7 Dec. 2012.
- 5) **S. Ozer**, D. Silver, K. Bemis, P. Martin, J. Takle, “Activity Detection for Scientific Visualization”, Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on, 117-118, 2011 (**Poster**).
- 6) **S. Ozer**, D. Silver, P. Martin, “Group Tracking in Scientific Visualization”, IEEE Visualization, Visweek, Providence, RI, 2011 (**Poster**).
- 7) **S. Ozer**, “An activity detection framework for 3D time varying scientific data sets”, ECE seminars, University of Massachusetts, Dartmouth, MA, May 2013 (**Presentation**).
- 8) **S. Ozer**, “Activity detection in scientific visualization”, VIVA seminars, University of Virginia, VA, August 2013 (**Presentation**).

REFERENCES

- [1] R. Adrian, C. Meinhart, C. Tomkins, "Vortex organization in the outer region of the turbulent boundary layer", *J. Fluid Mech.* 422, pp. 1–54, 2000.
- [2] J. K. Aggarwal and M. S. Ryoo, "Human activity analysis: A review", *ACM Computing Surveys*, 2010.
- [3] M. Albanese, R. Chellappa, V. Moscato, A. Picariello, V. S. Subrahmanian, P. Turaga, and O. Udrea, "A constrained probabilistic petri net framework for human activity detection in video," *IEEE Trans. Multimedia*, 2008.
- [4] E. Alpaydin, "Introduction to Machine Learning", MIT Press, Cambridge, MA, 2004.
- [5] N. Atmakuri, "Feature Tracking and visualization in Visit", M.Sc. thesis, Rutgers University, N.J., 2010.
- [6] H. Bay, A. Ess, T. Tuytelaars, L. V. Gool, "SURF: Speeded Up Robust Features", *Computer Vision and Image Understanding (CVIU)*, Vol. 110, No. 3, pp. 346--359, 2008.
- [7] K. G. Bemis, S. Ozer, G. Xu, P. A. Rona, D. Silver, "Event Detection for Hydrothermal Plumes: A case study at Grotto Vent", Abstract OS52A-05 presented at 2012 Fall Meeting, AGU, 3-7 Dec. 2012.
- [8] K. G. Bemis; G. Xu; J. Rabinowitz; P. A. Rona; D. R. Jackson; C. D. Jones: Understanding Plume Bending at Grotto Vent on the Endeavour Segment, Juan de Fuca Ridge, EOS, Transactions AGU, AGU Fall Meeting (abstract V11E-2544), 2011.
- [9] S. Bhattacharya, S. Habib, K. Heitmann, "Dark Matter Halo profiles of massive clusters: theory vs. observations", *Astrophysical Journal*, 2011.
- [10] R. Borgo, M. Chen, B. Daubney, E. Grundy, H. Jänicke, G. Heidemann, B. Höferlin, M. Höferlin, D. Weiskopf, and X. Xie, "A survey on videobased graphics and video visualization," *Eurographics (State of the Art Reports)*, Llandudno, UK, pages 1–23, 2011.
- [11] R.P. Botchen, S. Bachthaler, F. Schick, M. Chen, G. Mori, D. Weiskopf, T. Ertl, "Action-Based Multifield Video Visualization", in *Visualization and Computer Graphics*, IEEE Transactions on, 2008. 14(4): p. 885-899.
- [12] K. P.-T. Bremer, G. Weber, V. Pascucci, M. Day, and J. Bell, "Analyzing and Tracking Burning Structures in Lean Premixed Hydrogen Flames", *IEEE Transactions on Visualization and Computer Graphics* 16(2), 2010.
- [13] J. Caban, A. Joshi, and P. Rheingans. "Texture-based feature tracking for effective time-varying data visualization". *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1472–1479, 2007.
- [14] C. Castel, L. Chaudron, and C. Tessier, "What is going on? A high level interpretation of sequences of images," in *Proc. Workshop Conceptual Descriptions Images (ECCV)*, Cambridge, U.K., 1996, pp. 13–27.
- [15] J. Chen, D. Silver, and L. Jiang. "The feature tree: Visualizing feature tracking in distributed AMR datasets", In *Proceedings of IEEE symposium on Parallel and Large-Data Visualization and Graphics 2003*, pages 103–110, 2003.

- [16] M. Chen, D. Ebert, H. Hagen, R.S. Laramée, R. Van Liere, K.-L. Ma, W. Ribarsky, G. Scheuermann, D. Silver, "Data, Information and Knowledge in Visualization", IEEE Computer Graphics and Applications (IEEE CG&A), Vol. 29, No. 1, January/February 2009, pages 12-19.
- [17] S.-Y. Chen and X. Shan, "High-Resolution Turbulent Simulations Using the Connection Machine-2," Computers in Physics, vol. 6, no. 6, pp. 643-646, 1992.
- [18] D.J. Cook, N.C. Krishnan, P. Rashidi, "Activity Discovery and Activity Recognition: a New Partnership", Cybernetics, IEEE Transactions on (Volume:43 , Issue: 3), 2013.
- [19] T. Darom, Y. Keller, "Scale-Invariant Features for 3-D Mesh Models", Image Processing, IEEE Transactions on 21.5 (2012): 2758-2769.
- [20] R. David, H. Alla, "Petri nets & Grafcet," Prentice Hall, ISBN: 0-13-327537-X, 1992.
- [21] V.M. Fernandez, N.J. Zabusky, S. Bhat, D. Silver, and S.-Y. Chen, "Visualization and Feature Extraction in Isotropic Navier-Stokes Turbulence," Proc. AVS95 Conf., Boston, Apr. 1995.
- [22] K. S. Fu, & J. K. Mui, "A survey on image segmentation", Pattern recognition, 13(1), 3-16, 1981.
- [23] G. Gennari and G. D. Hager, "Probabilistic Data Association Methods in Visual Tracking of Groups.", IEEE CVPR, 2004.
- [24] A. Gezahegne and C. Kamath, "Tracking non-rigid structures in computer simulations", in Proceedings of the IEEE ICIP 2008.
- [25] N. Ghanem, D. Dementhon, D. Doermann, and L. Davis, "Representation and recognition of events in surveillance video using petri net," In CVPR Workshop, 2004.
- [26] M. Goebel, L. Gruenwald, "A survey of data mining and knowledge discovery software tools", SIGKDD Explor 1999, 1(1):20–33.
- [27] J. Han, Data Mining, "Concepts and Techniques", Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
- [28] The HDF Group. Hierarchical data format version 5, 2000-2010. <http://www.hdfgroup.org/HDF5>.
- [29] A. I. Hernandez, G. Carrault, F. Mora, L. Thoraval, G. Passariello, J. M. Schleich, "Multisensor fusion for atrial and ventricular activity detection in coronary care monitoring," IEEE Transactions on Biomedical Engineering, vol. 46, no. 10, pp. 1186–1190, 1999.
- [30] J. A. Insley; L. Grinberg; M. E. Papka; "Visualizing multiscale, multiphysics simulation data: Brain blood flow", Large Data Analysis and Visualization (LDAV), 2011.
- [31] Y.A. Ivanov, A.F. Bobick, "Recognition of Visual Activities and Interactions by Stochastic Parsing," IEEE Transactions on Pattern Analysis and Machine Intelligence, 22:852–872, 2000.
- [32] A. Jain, M. Murty, and P. Flynn, "Data clustering: A review," ACM Comput. Surv., vol. 31, no. 3, pp. 264–323, 1999.

- [33] G. Ji, H-W. Shen, and R. Wenger. "Volume tracking using higher dimensional isosurfacing". In *Proceedings of Visualization 2003*, pages 209–216, 2003.
- [34] G. Ji and H.-W. Shen, 2006. "Feature Tracking Using Earth Mover's Distance and Global Optimization," *Pacific Graphics 2006*.
- [35] S-W. Joo and R. Chellappa, "A multiple-hypothesis approach for multiobject visual tracking.", *IEEE Trans Image Process* 16(11):2849– 2854.
- [36] I. Koprinska, & S. Carrato, "Temporal video segmentation: A survey", *Signal processing: Image communication*, 16(5), 477-500, 2001.
- [37] C. Lakos, "From Coloured Petri nets to Object Petri nets", 16th international conference on applications and theory of Petri nets 1995, LNCS, Springer, pp:278-297, 1995.
- [38] D. Laney, P.-T. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci, "Understanding the Structure of the Turbulent Mixing Layer in Hydrodynamic Instabilities.", *IEEE Transactions on Visualization and Computer Graphics* Vol. 12, No. 5, pp. 1053-1060, 2006.
- [39] B. Lau, K. O. Arras and W. Burgard, "Tracking groups of people with a multi-model hypothesis tracker.", In: *International conference on robotics and automation (ICRA)*, Kobe, Japan.
- [40] G. Lavee, E. Rivlin, M. Rudzsky, "Understanding video events: a survey of methods for automatic interpretation of semantic occurrences in video," *Trans. Sys. Man. Cyber Part C* 39(5):489–504, 2009.
- [41] G. Lavee, M. Rudzsky, E. Rivlin, A. Borzin, "Video Event Modeling and Recognition in Generalized Stochastic Petri Nets," *IEEE trans. on Circuits and Systems for Video Technology*, Vol 20, N0:1, 2010.
- [42] J. N. K. Liu, K. Wang, Y.-L. He, X.-Z. Wang, "Formal Representation and Verification of Ontology Using State Controlled Coloured Petri Nets", *Reliable Knowledge Discovery*, pp 269-290, 2012.
- [43] D. G. Lowe, "Local feature view clustering for 3D object recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, Kauai, Hawaii (December 2001), pp. 682-688.
- [44] D. G. Lowe, "Object recognition from local scale-invariant features," *International Conference on Computer Vision*, Corfu, Greece (September 1999), pp. 1150-1157.
- [45] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, 60, 2 (2004), pp. 91-110.
- [46] J. G. Luhmann, S. C. Solomon, J. A. Linker, J. G. Lyon, Z. Mikic, D. Odstrcil, W. Wang, and M. Wiltberger, "Coupled model simulation of a Sun-to-Earth space weather event," *J. Atmos. Sol. Terr. Phys.*, 66, 1243, 2004.
- [47] K.-L. Ma, E. B. Lum, H. Yu, H. Akiba, M.-Y. Huang, Y. Wang, and G. Schussman. *Scientific discovery through advanced visualization. Journal of Physics: Conference Series*, 16(1):491, 2005.
- [48] C.Maes, T.Fabry, J.Keustermans, D.Smeets, P.Suetens, and D.Van-dermeulen, "Feature detection on 3-D face surfaces for pose normali- sation and recognition," in *Proc. 4th IEEE Int. Conf. BTAS*, Sep. 2010, pp. 1–6.

- [49] S. McKenna, S. Jabri, Z. Duric, A. Rosenfeld, and H. Wechsler, "Tracking groups of people", *Comput. Vis. Image Understanding*, vol.80, no. 1, pp. 42–56, 2000.
- [50] C. Muelder and K.-L. Ma. "Interactive feature extraction and tracking by utilizing region coherency". In *Proceedings of IEEE Pacific Visualization Symposium*, April 2009.
- [51] M. Mucientes and W. Burgard, "Multiple hypothesis tracking of clusters of people.", in *IEEE/RSJ international conference on intelligent robots and systems*, October 2006, pp 692–697.
- [52] T. Murata, "Petri nets: Properties, analysis and applications", *Proceedings of the IEEE* 77, no. 4 (1989): 541-580.
- [53] C. O'Farrell, M. P. Martin: Chasing eddies and their wall signature in DNS data of turbulent boundary layers, *Journal of Turbulence*, vol. 10, id. N15, 2009.
- [54] N. Oliver, B. Rosario, A. Pentland, "A Bayesian Computer Vision System for Modeling Human Interactions," *Proceedings of Intl. Conference on Vision Systems ICVS99*, Spain, January 1999.
- [55] M. L. Parry, P. A. Legg, D. H.S. Chung, I. W. Griffiths, M. Chen, "Hierarchical Event Selection for Video Storyboards with a Case Study on Snooker Video Visualization," *IEEE Visualization 2011 Proceedings*, 2011.
- [56] V. Pascucci, X. Tricoche, H. Hagen, J. Tierny (Eds.), "Topological methods in data analysis and visualization", ISBN 978-3-642-15014-2, Springer, 2011.
- [57] E. Palsson, H.G. Othmer, "A model for individual and collective cell movement in *Dictyostelium discoideum*", *Proc. Natl. Acad. Sci. USA* 97, 10448–10453, 2000.
- [58] M. Perše , M. Kristan , J. Perš , G. Mušič , G. Vučkovič , S. Kovačič, "Analysis of multi-agent activity using petri nets," *Pattern Recognition*, v.43 n.4, p.1491-1501, April, 2010.
- [59] J. L. Peterson, "Petri net theory and the modeling of systems", Prentice-Hall, ISBN: 0-13-661983-5, 1981.
- [60] C. A. Petri, "Kommunikation mit automaten", University of Bonn, West Germany, 1962.
- [61] R. Poppe, "A survey on vision-based human action recognition," *Image and Vision Computing*, 2010.
- [62] P. Rashidi, D. J. Cook, L. B. Holder, M. Schmitter-Edgecombe, "Discovering activities to recognize and track in a smart environment", *Knowledge and Data Engineering, IEEE Transactions on*, 23(4), 527- 539, 2011.
- [63] N. Ravi, N. Dandekar, P. Mysore, M. L. Littman, "Activity recognition from accelerometer data", In *AAAI*, pp. 1541-1546, 2005.
- [64] F. Reinders, F.H. Post, and H.J.W. Spoelder. "Visualization of time-dependent data using feature tracking and event detection", *The Visual Computer*, 17(1):55–71, 2001.

- [65] M. J. Ringuette, M. Wu, M. P. Martin, "Coherent structures in direct numerical simulation of turbulent boundary layers at Mach 3", *J. Fluid Mech.* 594, pp. 59-69, 2008.
- [66] P. A. Rona, K.G. Bemis, D. Kenchammana-Hosekote, and D. Silver, "Acoustic imaging and visualization of plumes discharging from black smoker vents on the deep seafloor", *IEEE Visualization 1998 Proceedings*, 1998, pp. 475-478.
- [67] R. Samtaney, D. Silver, N. Zabusky, and J. Cao, "Visualizing features and tracking their evolution", *IEEE Computer*, 27(7):20-27, July 1994.
- [68] K. Santilli, K. Bemis, D. Silver, J. Dastur, P. Rona, "Generating Realistic Images from Hydrothermal Plume Data", *IEEE Visualization*, pp: 91-98, 2004.
- [69] W. Schroeder, K. Martin, B. Lorensen, "Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition", ISBN-10: 193093419X, 2006.
- [70] P. Scovanner, S. Ali, and M. Shah, "A 3-dimensional sift descriptor and its application to action recognition," in *Proc. 15th Int. Conf. MULTI- MEDIA*, 2007, pp. 357-360.
- [71] D. Silver and X. Wang, "Tracking and visualizing turbulent 3d features", *IEEE Transactions on Visualization and Computer Graphics* 3,2, 129-141, 1997.
- [72] D. Silver and X. Wang, "Volume tracking", In *Proceedings of Visualization 1996*, 157-164.
- [73] D. Silver and X. Wang. "Tracking scalar features in unstructured datasets". In *Proceedings of Visualization 1998*, pages 79-86, 1998.
- [74] B.S. Sohn and C. Bajaj. "Time-varying contour topology". *IEEE Transactions on Visualization and Computer Graphics*, 12(1):14-25, 2006.
- [75] H. Storf, M. Becker, M. Riedl, "Rule-based Activity Recognition Framework: Challenges, Technique and Learning", *3rd International Conference on PervasiveHealth 2009*, pp 1-7, 2009.
- [76] E. M. Tapia, S. S. Intille, K. Larson, "Activity Recognition in the Home Using Simple and Ubiquitous Sensors," In *Pervasive Computing*, pages 158-175, 2004.
- [77] E. Trucco, E., & K. Plakas. Video tracking: a concise survey. *Oceanic Engineering, IEEE Journal of*, 31(2), 520-529 (2006).
- [78] P. Turaga, R. Chellappa, V.S. Subrahmanian, and O. Udrea, "Machine recognition of human activities: A survey," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 18, no. 11, pp. 1473-1488, Nov. 2008.
- [79] F.-Y. Tzeng and K.-L. Ma. "Intelligent feature extraction and tracking for large-scale 4d flow simulations". In *Proceedings of Supercomputing 2005 Conference.*, 2005.
- [80] Vizlab, Rutgers University, NJ, USA, "<http://vizlab.rutgers.edu>".
- [81] X. Wang, "Visualizing and tracking features in 3D time-varying datasets", Ph.D. Thesis, Rutgers University, 1999.
- [82] M. J. Waxman and O. E. Drummond, "A bibliography of cluster (group) tracking," in *Proc. SPIE Conf. Signal and Data Processing of Small Targets*, vol. 5428, 2004, pp. 551-560.

- [83] G. Weber, P.-T. Bremer, J. Bell, M. Day, and V. Pascucci, "Feature tracking using Reeb graphs", In Proceedings TopolnVis Workshop, 2009.
- [84] J. Wei, H. Yu, R. W. Grout, J. H. Chen, K. -L. Ma, "Visual Analysis of Particle Behaviors to Understand Combustion Simulations", IEEE Computer Graphics and Applications, 32(1): 22-33, 2012.
- [85] J. Wei, Z. Shen, N. Sundaresan, K.-L. Ma, "Visual Cluster Exploration of Web Clickstream Data", In IEEE Conference on Visual Analytics Science and Technology 2012.
- [86] D. Weinland, R. Ronfard, & E. Boyer, "A survey of vision-based methods for action representation, segmentation and recognition", Computer Vision and Image Understanding, 115(2), 224-241, 2011.
- [87] G. Xu, D. R. Jackson, K. G. Bemis, P. A. Rona, "Observations of the volume flux of a seafloor hydrothermal plume using an acoustic imaging sonar", Journal of Geochemistry, Geophysics, Geosystems, 2013.
- [88] R. Xu and D.I.I. Wunsch, "Survey of clustering algorithms", IEEE Trans. Neural Networks 16 (3) (2005) 645–678.
- [89] J. Yamato, J. Ohya, K. Ishii, "Recognizing human action in time-sequential images using hidden Markov model", In Computer Vision and Pattern Recognition, 1992. Proceedings of IEEE CVPR'92, pp. 379-385, 1992.
- [90] H. Yang, L. Shao, F. Zheng, L. Wang, & Z. Song, (2011). Recent advances and trends in visual tracking: A review. Neurocomputing, 74(18), 3823-3831.
- [91] A. Yilmaz, O. Javed, M. Shah, "Object tracking: A survey". Acm Computing Surveys (CSUR), 38(4), 13 (2006).

APPENDIX – I:

ATTRIBUTE GENERATION

Attributes play an essential role in segmentation, tracking and action recognition techniques. Recognition is typically done after computing several features for each object (the recognition process is equivalent to a retrieval or a matching process). The success of such recognition techniques fundamentally depends on whether a label can be assigned to a segmented (or detected) object based on its attributes. Such approaches inherently assume that the object attributes are distinctive so that the labels can be derived by discriminating the object attributes.

Consider a specific situation where each object is represented only with the mean and variance of its all member node values. Assume that two of these objects have the same mean and variance values such that $[m \ \sigma]^T$ (where m is the mean and σ is the standard deviation). In this case, if we know that one of these two objects represents a sphere and the other one represents an arrow, and if we use only the mean and variance values for such shape discrimination (i.e. recognition), regardless of the used technique, we would simply fail in deciding which object is a circle and which one is the arrow. Figure 45 illustrates this situation. In this example, a simple thresholding process could first segment (mask) the region of interests from the background. The next step would be deciding on which area (region of interest) is a sphere and which area is an arrow. Since both the sphere and arrow objects have a single (uniform)

value that is represented with the dark blue color, they both have the same mean and zero variance value. Therefore no approach would yield an accurate labelling for these two objects if we represent each of these objects by their individual mean and variance (or standard deviation) values only. This example shows the importance of attribute computation and selection for an efficient recognition process.

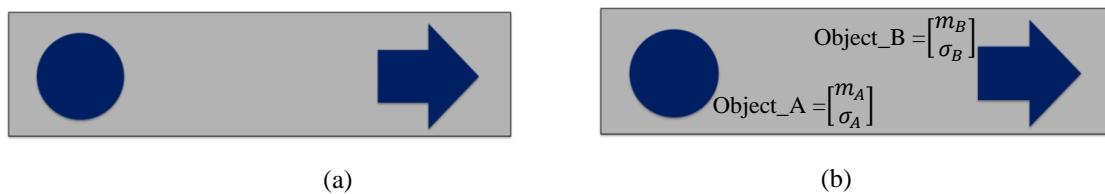


Figure 45: An illustration of two different objects with the same mean and variance values. (a) An image containing a sphere and an arrow with a uniform value (represented with the dark blue color), (b) representing each object with its mean and standard deviation values.

In many computer vision applications, the recognition process is highly (and sometimes solely) depends on the shape based attributes, i.e. the shape descriptors. Shape descriptors can be categorized under two labels namely global descriptors and local descriptors.

Global attributes: Global attributes are the attributes that “summarize” a unique characteristic of an object. Examples are mean, variance values of all the node values of an object, volume, mass, moments, etc. All these attributes summarize the entire set of nodes (voxel or pixel) of an object or (or its mesh points) with a less and usually fixed number of values. Therefore we can say that these features map M number of voxels (where M varies from an object to another) into N number of attributes (where N is fixed for all the objects). Usually (but not necessarily) $M > N$. An example for this case is the mean value

computation which is a mapping from many nodes to one value, i.e., one attribute.

Local attributes: Local attributes are the individual points of an object that can describe the object uniquely when considered all together. These special points are sometimes called *interest points* since they can (collectively) discriminate the object from other types. They are computed from only a small portion of the entire data. Example local feature types are the corners (and their numbers), local curvatures, and local histograms. The entire set of local attributes (altogether) also summarizes a characteristic of an object. Since the number of such interesting points changes from object to object, the number of the local attributes varies from object to object. On the contrary, the length (the number) of computed attributes is fixed in global features for each object. *Local attributes* are mostly used for shape related tasks and therefore also named *shape descriptors*.

Various techniques have been proposed to “detect” the informative (salient) local points of an object for different data types. The data type can be one dimensional (1D), two dimensional (2D) or three dimensional (3D) based on the data acquisition (or generation) environment, its geometrical and topological structure.

Recently, many researcher focused on first defining and then detecting the local attributes due to the success of the Scale-Invariance Feature Transform (SIFT) in many computer vision applications including recognition and image retrieval. Following the success of SIFT, Speeded Up Robust Features (SURF)

algorithm also has been proposed for computer vision applications. While there are some technical limitations in these algorithms, there are also legal limitations since both algorithms are currently patented to their respective owners. Currently there are many available techniques that extract local descriptors based on different criteria. Examples are Local Energy based Shape Histogram (LESH), Gradient Location and Orientation Histogram (GLOH).

In order to understand the local descriptor concept better, we will go over the SIFT algorithm as a representative algorithm here.

A1.1. Scale-invariance feature transform (SIFT)

Local attributes were proposed and studied before the SIFT algorithm. However, SIFT algorithm emphasized on their importance with its robust and successful image analysis, recognition, and retrieval applications. The fundamentals of SIFT algorithm can be found in the papers [43], [44] and [45].

According to [45] SIFT algorithm has four major steps:

- 1) Scale-space extrema detection: This step finds (detects) all the extrema points in *scale space* as candidate local attribute points.
- 2) Keypoint localization: For each candidate local attribute point (*keypoint*) found in step 1 (in *scale space*), compute the actual location and scale values.
- 3) Orientation Assignment: For each keypoint, compute the *orientation* and *magnitude* values based on the local *image gradients*.
- 4) Keypoint Descriptor: Compute (derive) a set of attributes for each keypoint based on the neighbour pixels.

A1.2. Scale space extrema detection

Keypoint locations that are invariant to scale changes of the image can be detected by finding stable points across all possible scales [45]. A scale is a filtering parameter (a distance measure in image filtering). A discrete scale space of an (2D) image can be considered as a discrete 3D volume where the third dimension is formed of by filtering the original image with a constant increment in scale. A sampled illustration of a scale space is shown in Figure 46.



Figure 46: Illustration of the concept scale in scale space. (a) The original image, (b) samples taken from a scale space of the original image. As it is shown, the filtering process removes the high frequency components of the image and the result is a blurred version of the original image at a certain scale.

The most intuitive and appropriate filter is Gaussian kernel (which is a space-space kernel) according to [45]. Therefore, a scale space of an image is formed by convolving the Gaussian kernel with the image at different and continuous scales.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad (9)$$

Where $I(x, y)$ is the pixel value of the image I at the (x, y) coordinate, $*$ is the convolution operation, $L(x, y, \sigma)$ is the filtered (scaled) version of the original image I at the scale σ . $G(x, y, \sigma)$ is the Gaussian kernel:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} \quad (10)$$

It is well known that the Laplacian of a Gaussian operator is good at detecting edges and blobs. As we smooth and down-sample an image, we get rid of the sharp corners existing in the image, thus more objects gets blob-like shapes. Therefore if we apply the Laplacian of a Gaussian operator ($\nabla^2 G$) on an image at different scales, then we could detect edges and blobs at different scales. Since we already computed scale space image at various scales, the Laplacian of a Gaussian operator can be approximated by using two consecutive samples of L in scale.

The partial derivative of Gaussian can be expressed as:

$$\frac{\partial G(x, y, \sigma)}{\partial \sigma} = \sigma \nabla^2 G(x, y, \sigma). \quad (11)$$

On the other side, the partial derivative of a Gaussian kernel can also be approximated by using the finite difference approximation. As a result, the Laplacian of a Gaussian kernel (G) can be approximated as:

$$\sigma \nabla^2 G(x, y, \sigma) = \frac{\partial G(x, y, \sigma)}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \quad (12)$$

Where k is a constant such that for a given pair of consecutive scales σ_1 and σ_2 (where $\sigma_1 < \sigma_2$) $\sigma_2 = k\sigma_1$. Therefore the difference of two Gaussians at consecutive scales can be written as:

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G(x, y, \sigma). \quad (13)$$

Since the Gaussian kernel is also used to create a scale space of an image, the approximation of Laplacian of a Gaussian applied on an image, can be computed by using the samples from the scale space of the image. Applying the Laplacian of a Gaussian on an image can be approximated as the difference of two consecutive scale space images since:

$$D'(x, y, \sigma) = \nabla^2 G(x, y, \sigma) * I(x, y) \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{\sigma^2(k - 1)} * I(x, y) \quad (14)$$

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \approx \sigma^2(k - 1)D'(x, y, \sigma) \quad (15)$$

Where $D'(x, y, \sigma)$ is the actual result and $D(x, y, \sigma)$ is its approximation. As a result, the Laplacian of a Gaussian operator (filter) is approximated by the difference of two Gaussians (DoG). Since the discrete scale space of an image is already computed at various scales, the difference of scale space images are used in SIFT such that:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma). \quad (16)$$

$D(x, y, \sigma)$, i.e. the difference of Gaussians (DoG), is computed for each consecutive pairs in a discrete scale space. Once DoG images (samples) are constructed, the next step is finding the minimum and maximum points. These are the extrema points that appear or disappear along the scale at a certain location. Lowe suggests using three DoG samples taken at 3 consecutive scales

(such as at the scales σ_1 , $\sigma_2=k\sigma_1$ and $\sigma_3=k\sigma_2$) to find the minimum and maximum points. This process is shown in Figure 47 [45].

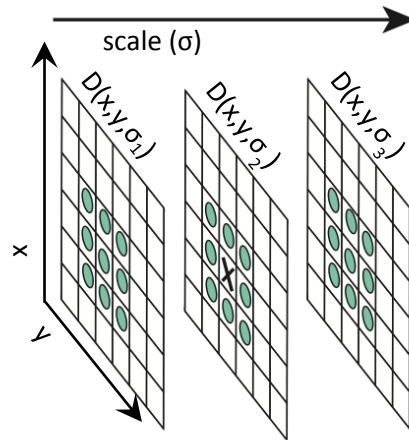


Figure 47: Computing the local extrema points. The pixel denoted “X” on the picture, is a local extremum point if it is the minimum or the maximum among all the green points in DoG samples (images), image source: [45].

The value at the location X (shown in Figure 47) is compared to its 8 neighbors in $D(x,y,\sigma_2)$, its 9 neighbors in $D(x,y,\sigma_1)$ and its 9 neighbors in $D(x,y,\sigma_3)$. If its value is greater than all of its neighbor values then it is saved as a local maximum. If it is smaller than all its neighbors, then it is saved as a local minimum point.

Once all the extrema points are found in a scale space, a new octave is created by downsampling the original image. Down-sampling is done by taking every other pixel along x and y dimensions. And DoG images of this down-sampled image are computed and then the extrema points are also computed in this octave. An octave is the discrete scale space of an image. The scale space computed by using the original size is considered as the first octave, and second octave is the discrete scale space created from the down-sampled version of the

original image. In each octave, the image dimensions are reduced to half. An illustration of various octaves of a given image is shown in Figure 48.

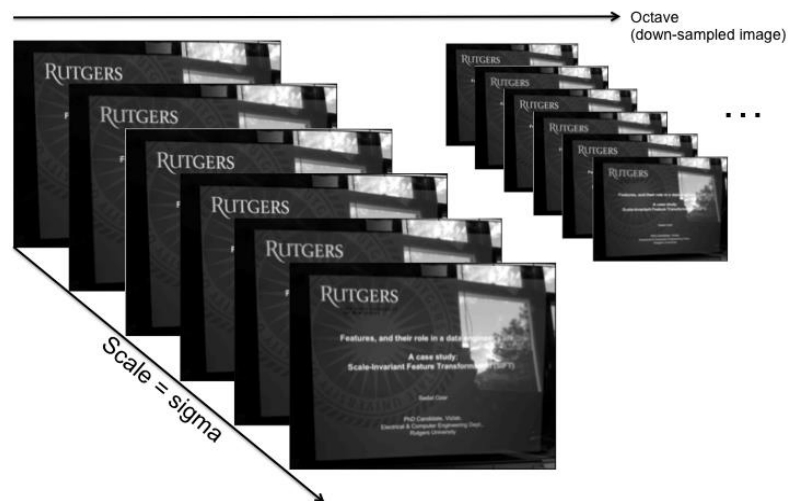


Figure 48: Visualizing the first and second octaves of a given image.

Summary:

- 1) Compute the scale space of an image via the convolution process,
- 2) Construct the DoG images for each consecutive pairs (along the scale) in scale space,
- 3) Find the local minimum and maximum points in DoG images according to the Figure 47.
- 4) Down-sample the image,
- 5) Repeat step 1 through 3 for the down-sampled image.

Notice that since image is down-sampled, the scale σ , is the two times of the σ used in the previous octave. This process ensures that the distance value remains the same in filtering in each octave.

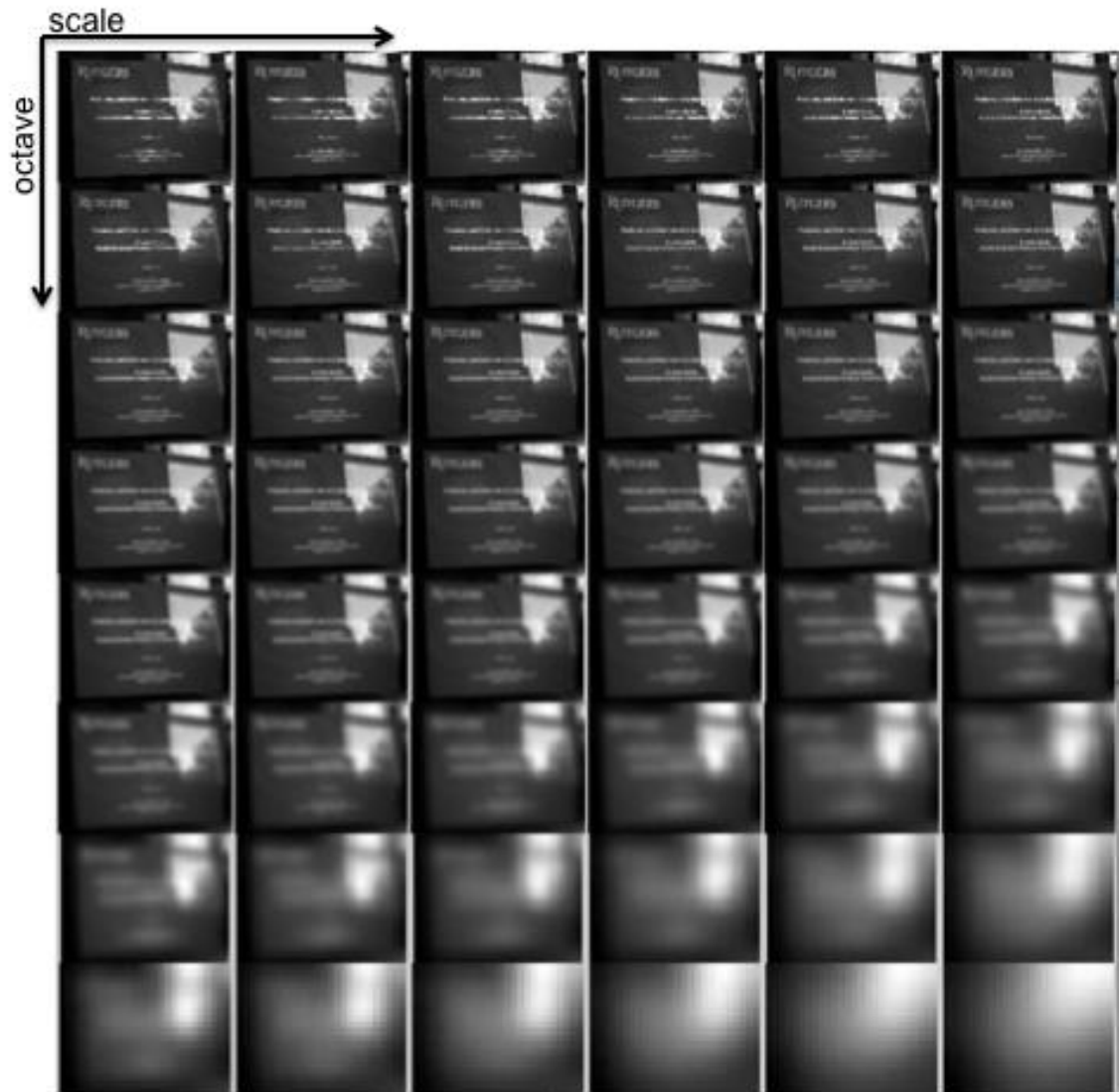


Figure 49: The effect of scaling at various octaves and scales. The top left image is the original image.

Figure 49 shows samples taken from scale space along both scale and octave. The original image (the image shown in top left corner in Figure 49) is a high definition image with the dimensions ~ 1000 by 1000 . Therefore, when it is resized to fit the figure, it is distorted. However, in its actual size, it looks crispier than any other image shown in the figure.

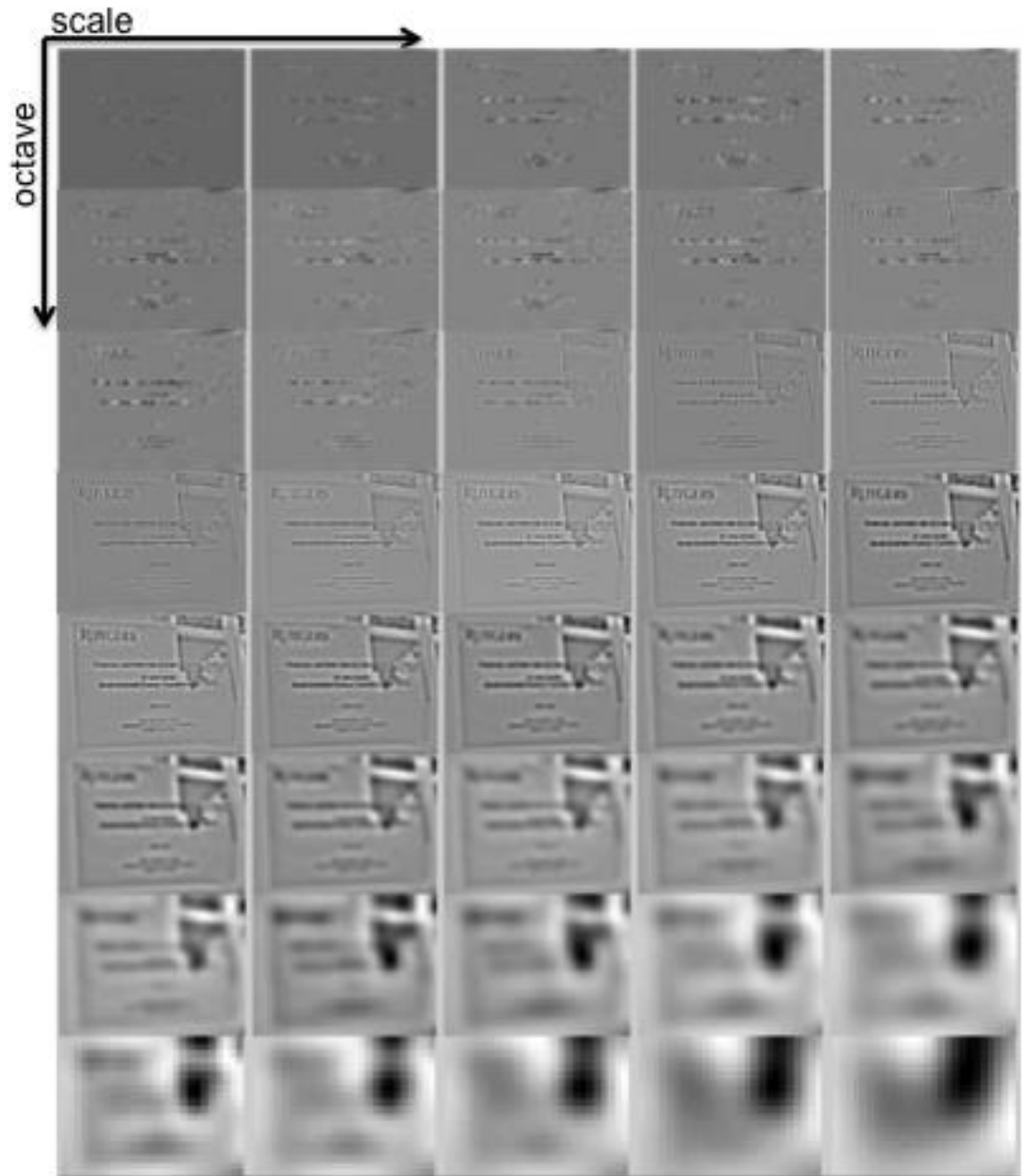


Figure 50: Computed DoG images for Figure 49.

A1.3. Keypoint localization

Keypoints are detected at different scales and octaves. It is essential to find the most accurate location for those points if they are detected in down-sampled images. This process becomes more important as the number of octave increases since the difference between two neighbour pixels change as the image is downsampled from one octave to the next.

Lowe suggests approximating $D(x,y,\sigma)$ by using only the first three terms of its Taylor expansion where the function is shifted to make the origin at the sample point. Then

$$D(\mathbf{x}) \approx D + \frac{\partial D^T}{\partial \mathbf{x}} \mathbf{x} + \frac{1}{2} \mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2} \mathbf{x}. \quad (17)$$

Where D and its derivatives are evaluated at the sample point $\mathbf{x} = [x,y,\sigma]^T$. Since we are interested in finding the local minimum and local maximum points (and since the derivative of $D(\mathbf{x})$ is zero at local min and max points), we can simply take the derivative of this approximation with respect to \mathbf{x} and equate that to zero to estimate the local minimum (or local maximum) location $\hat{\mathbf{x}}$ as:

$$\hat{\mathbf{x}} = - \frac{\partial^2 D^{-1}}{\partial \mathbf{x}^2} \frac{\partial D}{\partial \mathbf{x}} \quad (18)$$

Then the difference value at the estimated location is:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2} \frac{\partial D^T}{\partial \mathbf{x}} \hat{\mathbf{x}} \quad (19)$$

Since $\hat{\mathbf{x}}$ is an offset (the relative distance) from the actual pixel coordinate, it should be close to the original pixel. That is only true if $\hat{\mathbf{x}} < 0.5$. Otherwise, it will be closer to another pixel. Therefore, this operation should be re-computed for

that new (closer) pixel. Lowe suggests that if the value of $|D(\hat{x})| < 0.03$, the keypoint at that location should be deleted from the keypoint list.

Since the Laplacian of a Gaussian is good at detecting the edges and blob like structures, the algorithm will detect many keypoints along the edges. This will yield an increased number of keypoints along the edges. To eliminate such keypoints, SIFT algorithm computes Hessian matrix \mathbf{H} to estimate the local (principal) curvatures of $D(x,y,\sigma)$.

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \quad (20)$$

The curvature information of $D(x,y,\sigma)$ at the point (x,y) can be obtained by investigating the Eigen values of \mathbf{H} . Assume that two Eigen values of \mathbf{H} are e_1 and e_2 . Then the sum of e_1 and e_2 is equal to Trace of \mathbf{H} , and the determinant of \mathbf{H} is equal to the multiplication of the two Eigen values, i.e.,

$$\text{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = e_1 + e_2 \quad (21)$$

$$\text{Det}(\mathbf{H}) = D_{xx}D_{yy} - D_{xy}D_{xy} = e_1e_2 \quad (22)$$

If the determinant is negative, the point is deleted from the keypoint list. Assume that $e_1 > e_2$ and $e_1 = re_2$, then

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} = \frac{(e_1 + e_2)^2}{e_1e_2} = \frac{(re_2 + e_2)^2}{re_2e_2} = \frac{(r+1)^2}{r} \quad (23)$$

Then we need to check if $\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$.

In SIFT, for $r=10$, if the point does not satisfy the above inequality, then it will be ignored.

A1.4. Orientation assignment

Until now, we only found “interesting points”, i.e. keypoints in the image. That is, we only have a coordinate (x,y) information of some “selected” pixels where the selection is based on the keypoint localization process described above. Next step is computing rotation invariant, and scale invariant attributes that could uniquely characterize these selected points. For that purpose, SIFT computes a set of 128 attributes for each keypoint. Since each set describes a keypoint, they are called descriptors. SIFT uses local gradient magnitudes and orientations within the neighborhood of each keypoint to form keypoint’s descriptors. Since orientation is related to rotation, SIFT suggests to remove a reference orientation from these computed attributes. Thus the orientation based attributes become rotation invariant. In SIFT a gradient orientation is defined as:

$$\theta(x, y) = \tan^{-1} \left(\frac{L(x, y+1, \sigma_k) - L(x, y-1, \sigma_k)}{L(x+1, y, \sigma_k) - L(x-1, y, \sigma_k)} \right) \quad (24)$$

The reference orientation is determined by forming a histogram of neighbour orientations. The gradient magnitude and its orientation values are computed within the $1.5\sigma_k$ distance of the keypoint, where σ_k is the scale for the current octave. Before forming the histogram, to consider the closer pixels more than the further away points, the magnitudes are weighted with a Gaussian circular window with a $\sigma = 1.5\sigma_k$. where the magnitude of the gradient is defined as:

$$m(x, y) = \sqrt{(L(x+1, y, \sigma_k) - L(x-1, y, \sigma_k))^2 + (L(x, y+1, \sigma_k) - L(x, y-1, \sigma_k))^2}$$

Once all the orientations and magnitudes are computed within the neighbourhood of the keypoint, all the orientations (angles) are quantized (binned) into 36 bins covering 360 degrees. After the quantization, each histogram value (for each bin) is computed by summing the magnitudes of the neighbour pixels whose orientations fall into the corresponding bin. Once the histogram is formed, the highest peak is chosen to assign an orientation to the keypoint. For that, in order to decrease the effects of the quantization (binning) process (i.e, for a better orientation estimate), SIFT fits a parabola to the 3 histogram values closest to each peak. This final value is assigned to the keypoint as orientation. If there are more than one peaks that are greater than 80% of the highest peak in the histogram, then each of these points also used to generate a new keypoint at the same location.

A1.5. Keypoint descriptors

So far we detected keypoint locations, and assigned them an orientation. These variables are not invariant to affine transformations. SIFT suggests using local gradients and magnitudes within a region to form a histogram. The histogram values are the final keypoint descriptors.

The histogram values (descriptors) are computed by first forming n by n subregions in an m by m neighborhood where $m \geq n$ and the keypoint is centered in the m by m region. (SIFT suggests creating 4 by 4 subregions in the 16 by 16 neighborhood of each keypoint). Each neighbor's gradient orientation and magnitude is computed and these magnitudes are weighted by a Gaussian window with a $\sigma = 1.5\sigma_k$. The points outside the Gaussian window are ignored.

The computed orientations are rotated relative to the keypoint's orientation. This process makes the computed orientations invariant to the rotation operations.

The process of computing the descriptors of a keypoint are illustrated in Figure 51 [45].

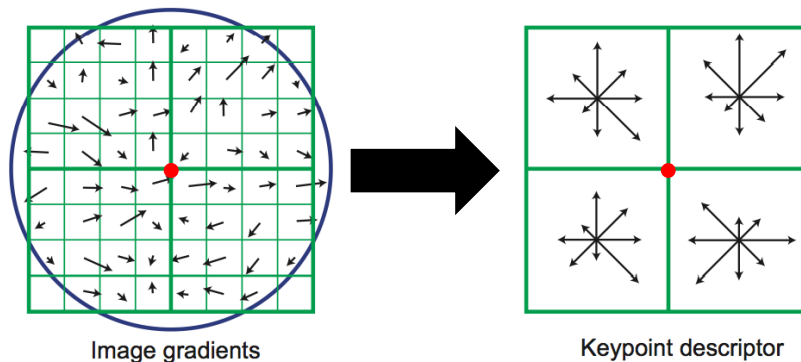


Figure 51: Keypoint descriptor computation process. First a Gaussian window is places where the center is the keypoint (red point). Then each neighbor pixel's magnitude and orientation is computed and magnitudes are weighted with the Gaussian window weight. Then the orientations are quantized and histograms for each subblock (In this figure for each 4 by 4 sub-block). Image origin: [45].

For descriptor computation, each orientation is quantized into 8 bins (as opposed to 36 bins for the reference orientation computation). Since SIFT computes 4 by 4 arrays around the keypoint, it computes these 8 bins for each block within the 4 by 4 array. As a result, $4 \times 4 \times 8 = 128$ elements are computes as the attribute vector for each keypoint. Once the vector is formed, it is normalized to the unit length. Once the attribute vector is normalized, the values above 0.2 are truncated to 0.2.

A1.6. 3D-SIFT and SIFT-like algorithms

Original SIFT algorithm is developed mainly considering the problems associated with 2D image (data) sets (such as segmentation, image recognition, retrieval and affine transformation). Therefore all the operations are defined and used in 2D. In many data sets (Such as medical scans, videos) the data is considered 3D and, hence, a 3D version of the SIFT algorithm would be more efficient than applying the original SIFT algorithm on 3D data. Therefore Scovanner et. al. [69] proposed a 3D SIFT algorithm. 3D SIFT extends the 2D equations of the original SIFT algorithm into 3D. However, since many 3D applications require (or operate on) mesh structures, there are different mesh based SIFT algorithms proposed. These algorithms compute SIFT-like features from mesh structures. Examples are MeshSIFT can be found in [48] and [19].

While the results are not presented in this dissertation, our preliminary tests for applying meshSIFT on scientific objects did not show promise for tracking scientific objects. This is due to the blob-like structure of scientific objects in general. However, the application areas of SIFT or MeshSIFT-like algorithms are broader. Future work may include using such algorithms for modelling complex shape-based actions or activities of scientific objects.

APPENDIX – II:

USER MANUAL

This chapter describes the content of the both the output files and the configuration files for the existing implementations of the proposed activity detection framework used in this dissertation. We will first describe the content of the Feature & Group tracking implementation and then will describe the configuration file for the Petri Net implementation.

Content of the configuration file for *feature & group tracking implementation* and the meaning of the used keywords are given below:

DATA_FILES_PATH: This is the path of the file starts and ends with “/”. Example usage is given in Figure 52.

GENERATED_FILES_PATH: This is the path under which all the created files will be saved. Example usage is given in Figure 52.

FILE_BASE_NAME: This is the unchanging part of the name of the data files. For example, assume that a simulation data set is formed of following 5 files: *nwa_avg_1.nc*, *nwa_avg_2.nc*, *nwa_avg_3.nc*, *nwa_avg_4.nc* and *nwa_avg_5.nc*. In this case, the data at each timestep is saved in each file and **FILE_BASE_NAME** (unchanging part) in this data set is: *nwa_avg_*.

FILE_EXTENSION: In the above example all files end with the extension “.nc”. Therefore in the above simulation data set, this variable is set to .nc.

INITIAL_TIME_STEP: This is the time index that the data set starts with. For example in the above simulation data set, the data set starts from the first timestep (nwa_avg_1.nc). Therefore for that example, initial timestep is 1.

FINAL_TIME_STEP: This is the time index that the data set ends with. For example in the above simulation data set, the data set ends with the data set nwa_avg_5.nc. Therefore for that example, the final timestep is 5.

TIME_STEP_INCREMENT: This variable defines how the time index (timestep) changes (the difference) from one file to the next. For example, in the above simulation data set, the indices increase one by one. Therefore timestep increment in this case is 1.

TIME_STEP_PRECISION: This variable defines in which format the time index (timestep) is saved. For example, if the index part is saved as 001, 002, 003, 004, 005, then the timestep precision would be 3. If the timestep indices are saved as 0001, 0002, 0003, 0004, 0005, then the timestep precision would be 4. In the above simulation data set, the indices are saved in single digits. Therefore timestep precision in this case is 1.

VARIABLE_NAMES: If there are more than one variable saved in the data file, this variable defines which variable (or variables) to be loaded in the data. An example is given in Figure 52.

THRESHOLD1: This is the threshold value that is used for region growing algorithm. The region growing algorithm could segment the region of interests based on whether the values are smaller or greater than this threshold (the default

value is segmenting the greater regions in the data. However, the smaller regions can also be segmented by a change in the code).

THRESHOLD2: This variable is saved here for future references. In the future, two or more than two variables can be used at once to segment a region of interest.

DELTA_X_THRESHOLD: This is the distance based threshold along x axis that is used in packet extraction and tracking algorithm. If the purpose is feature tracking, this value must set to 0 along with Delta y and delta z thresholds.

DELTA_Y_THRESHOLD: This is the distance based threshold along y axis that is used in packet extraction and tracking algorithm. If the purpose is feature tracking, this value must set to 0 along with Delta x and delta z thresholds.

DELTA_Z_THRESHOLD: This is the distance based threshold along z axis that is used in packet extraction and tracking algorithm. If the purpose is feature tracking, this value must set to 0 along with Delta y and delta x thresholds.

SMALLEST_OBJECT_VOLUME_TO_TRACK: This is the threshold to filter small objects out. The integer number set for this variable is the minimum volume (where the volume is the total number of the voxels or nodes in a segmented feature). If the volume of the object is smaller than this value, it will be ignored and will not be saved as a feature.

X_Dim: This is the integer value that defines the total number of voxels (i.e., the dimension along the x axis in the data).

```
DATA_FILES_PATH: /Users/data/
GENERATED_FILES_PATH: /Users/data/GENERATED_TRACK_FILES1/
FILE_BASE_NAME: nwa_avg_
FILE_EXTENSION: .nc
INITIAL_TIME_STEP: 1
FINAL_TIME_STEP: 5
TIME_STEP_INCREMENT: 1
TIME_STEP_PRECISION: 1
VARIABLE_NAMES: omega
THRESHOLD1: -0.00000000000331
THRESHOLD2: 40.6
DELTA_X_THRESHOLD: 0.01
DELTA_Y_THRESHOLD: 0.01
DELTA_Z_THRESHOLD: 0.01
SMALLEST_OBJECT_VOLUME_TO_TRACK: 15
X_Dim: 722
Y_Dim: 362
Z_Dim: 40
X1_Dim: 721
Y1_Dim: 361
Z1_Dim: 39
X0_Dim: 0
Y0_Dim: 0
Z0_Dim: 0
```

Figure 52: A sample configuration file for feature and group tracking module.

Y_Dim: This is the integer value that defines the total number of voxels (i.e., the dimension) along the y axis in the data.

Z_Dim: This is the integer value that defines the total number of voxels (i.e., the dimension) along the z axis in the data.

The below variables allow user to select a portion of the data and process only that selected portion (i.e., sub-region) in the data. A portion of the data can be defined by the coordinates of two points (Namely Point1 and Point0). These two points define the extends of the selected portion. They can be virtually visualized as the lower left corner (Point0) and the upper right corner (Point1) of a 3dimensional box. Each point can be defined by their coordinates in Cartesian coordinates.

X1_Dim: This is an integer index defining the x coordinate of the point 1 (Point1) that define the most right point that should be used in the data (if the entire data needs to be processed, than this variable must be set to X_Dim-1).

Y1_Dim: This is an integer index defining the y coordinate of the point 1 (Point1) that define the most right point that should be used in the data. (if the entire data needs to be processed, than this variable must be set to Y_Dim-1).

Z1_Dim: This is an integer index defining the z coordinate of the point 1 (Point1) that define the most right point that should be used in the data. (if the entire data needs to be processed, than this variable must be set to Z_Dim-1).

X0_Dim: This is an integer index defining the x coordinate of the point 0 (Point0) that define the least left point that should be used in the data. (If the entire data needs to be processed, than this variable must be set to 0).

Y0_Dim: This is an integer index defining the y coordinate of the point 0 (Point0) that define the least left point that should be used in the data. (If the entire data needs to be processed, than this variable must be set to 0).

Z0_Dim: This is an integer index defining the z coordinate of the point 0 (Point0) that define the least left point that should be used in the data. (If the entire data needs to be processed, than this variable must be set to 0).

Content of the configuration file for Petri Net implementation is shown in Figure 53. The keywords used in this file are described below.

TrackingFileName: This variable defines the location and name of the tracking file.

IsTrackingInfoProvided: This variable is reserved for future improvement. This variable is used to state if tracking information is necessary or if it is inherent in the segmentation (attributes file). The value of this variable is Boolean and is provided as YES or NO. The current implementation always assumes that the tracking information is necessary and is provided. However, since in some applications there may be only one object, or the order of the objects remains the same in the attribute files at each timestep, a specific tracking step may not be necessary. Therefore, the term NO is reserved to describe this situation in future implementations.

```

TrackingFileName = Users/ GENERATED_TRACK_FILES1/nwa_avg_1.trakTable
IsTrackingInfoProvided = YES
InitialFrameNumber = 1
FinalFrameNumber = 14
Variables = F0D6
Places = (2>1); (2>1) ; (2>1)
Transitions = (F0D6>1) ; (F0D6>2) ;
FilePath = /Users/NewParsedData/
TransitionsFileBaseName = nwa_avg_
TransitionsFileExtension = .trak
PlacesFileBaseName = nwa_avg_
PlacesFileExtension = .trak
DefaultActionsFileBaseName = nwa_avg_
DefaultActionsFileExtention = .trak
inputarcs = 1 1 1 2 2 1
outputarcs = 2 1 1 3 2 1
initialMarkings = 0 0 0
FinalPlaceIDs = 3
InitialPlaceIDs = 1

```

Figure 53: A sample configuration file for the Petri Net module.

InitialFrameNumber: This is an integer number describing the starting time index.

FinalFrameNumber: This is an integer number describing the final timestep index that should be processed in the data set.

Variables: These are the Petri Net variables that are used to define the conditions in both transitions and places.

Places: This is the line that defines the conditions for each place. Each place condition is separated from other place conditions by ";". The number of conditions is equal to the number of the places in the Petri Net. The first condition is associated to the first place, the second condition is associated to the second place and so on, in the Petri Net.

Transitions: This is the line where all the transition conditions are defined. Each transition condition is separated from other transition conditions by ";". The total number of transitions in the Petri Net is equal to the number of total transitions. The condition order describes the transition index. I.e., the first condition is associated to the first transition; the second condition is associated to the second transition and so on, in the Petri Net.

FilePath: This is the path describing the folder where all the input files (including attributes or actions of objects) are located.

TransitionsFileBaseName: This is the base name of the data set (i.e., the part of the file name that does not change by time). Our implementation assumes that the object attributes are saved in separate timesteps. And for each timestep, all the object attributes are listed in a single file. Our implementation allows to define different file names for transitions and for places. The assumption is that, all the transition files start with the same constant name. The full file name is:

“FilePath+ TransitionsFileBaseName+ currenttimeindex+ TransitionsFileExtension”.

TransitionsFileExtension: This is the file extension for the attributes used in transition conditions. (as described in TransitionsFileBaseName).

PlacesFileBaseName: This is the file base name for the Place attributes (similar to the TransitionsFileBaseName).

PlacesFileExtension: This is the file extension for the attributes used in place conditions.

DefaultActionsFileBaseName: This is the action file’s base name. The file defines the name of the “default” actions. (Such as merge, split or continuation.)

DefaultActionsFileExtention: This is the extension of the action files.

Inputarcs: Input arcs are the arcs from a place to a transition. Each arc is described by a triplet. A triplet is formed of 3 integers where the first integer is the PlaceID, the second intereger is the TransitionID and the third integer is the arc weight. While the token-tracking Petri Nets assume that arc weights are 1, for future improvements, we keep user to specify arc weights here.

Outputarcs: An output arc is the arc that originates from a transition and ends at a place. Similar to inputarcs, each outputarc is defined by a triplet. Each triplet represents a single output arc by 3 integers. The first integer represents the PlaceID, the second integer represents the TransitionID, and the third integer represents the arc weight.

```

-----
object 0 attributes:
Max position: (287.232605, 19.894339, -0.010823) with value: -0.000000
Node #: 9156671
Min position: (287.232605, 19.894339, -0.066988) with value: -0.000000
Node #: 3145299
Integrated content: -0.000000
Sum of squared content values: 0.000000
Volume: 44
Centroid: (287.223584, 19.895294, -0.054405)
Moment: Ixx = 0.000742
Iyy = 0.000008
Izz = 0.000454
Ixy = -0.000079
Iyz = -0.000010
Izx = 0.000099
-----
object 1 attributes:
...
...

```

*Figure 54: A sample from the content of an *.attr file.*

initialMarkings: While the initial marking is always assumed to be zero (i.e., the total number of tokens in each place is zero initially) at the beginning in our current implementation, this keyword reserved here to be used in future implementations.

FinalPlaceIDs: These are the set of PlaceIDs that are considered important to the user, or that are the end of an activity, or multiple activities. For example, two different activities can be modeled in a single Petri Net, and therefore there would be two different final places in the model (at least).

InitialPlaceIDs: These are set of Places in which a new token can enter the Petri Net as a starting place. There can be multiple initial places in a single activity model.

The output files of the standalone feature & group tracking algorithm are given below. The output files of the feature and group tracking implementation are saved in separate files in each timestep. Most of these files start with the data name. They are:

- datanameX.attr
- datanameX.group
- datanameX.poly
- datanameX.trak
- datanameX.trakTable
- datanameX.uocd
- dataname_comp_1_END.list

- t.groupTrakTable
- ColorMapX.txt

The the letter X in the above listed names refer to the ecah timestep. Therefore these files (except the .trakTable, t.groupTrakTable and the dataname_comp_1_END.list files) are generated separately for each time step.

A sample from the content of the datanameX.attr files is given in Figure 54. This file lists the computed attributes for each object in a human readable form. Objects are given an integer ID starting from 0. Max position is the position of the maximum value within the feature. Node number is the node ID within the datafile. Integrated content is the sum of all the nodal values within the feature. Volume is the total number of nodes within the feature. Centroid is coordinate of the geometric centroid. The lxx, lyy, lzz, lxy and lyz are the elements of the 2nd order moment (tensor matrix). These values are listed for each individual feature extracted within the timestep. A detailed description of each of these terms can be found [81].

A sample from the content of the datanameX.group file is given in Figure 55. In Figure 52, each group is listed in one line. Each line starts with the group ID and then the member feature IDs are listed. For example the first line in the group file are “ 1 6 16” means groupID 1 has the features 6 and 16 as members. Similarly, Group 2 has feature 17 and feature 18 as members. Group 3 has only one member (feature 22). While the earlier versions include more attributes in this file, the most recent version is limited to include only the member feature IDs

due to the size constraints. Most group attributes can be computed by combining the feature attributes with the information available in the group files.

```

1  6  16
2  17 18
3  22
4  9  27 28
5  30
6  32
...
...
```

*Figure 55: A sample from the content of an *.group file.*

The content of the .poly, .trakTable and the uocd files remain the same as in [81] and in [5]. The .trak files compute the feature attributes and save them in a more compact way than the .attr files. The content and the order in this file changes from one version to the next. The content of the most recent version created by the standalone implementation is given below as example. In track files, the objects are listed according to their IDs starting from 0. That is, the attributes listed in the first line belongs to the feature 0, the second line belongs to the feature 1, etc. Each line contains the following 20 attributes in the order of:

```

mass,          volume,          centroid[x,y,z],          boundingboxcoordinates(
lowerleftcornercoordinates[x,y,z],          upperrightcornercoordinates[x,y,z]),
minimumvalueLocation[x,y,z],          maximumvalueLocation[x,y,z], minimumValue,
```

maximumValue, GroupID. An example file content is given in Figure 56 for the first two features.

```

20.976601  527  6.476810 -0.612426  0.022417  6.421410 -0.645774
0.002389  6.512880 -0.564202  0.074619  6.512880 -0.632179  0.033016  6.458000
-0.638976  0.030187  0.014757  0.139331  55

2.183900  100  6.976500 -0.327972  0.010872  6.951950 -0.346679
0.003298  7.006840 -0.299095  0.027541  6.970250 -0.319488  0.003298  6.970250
-0.333083  0.010232  0.014932  0.035414

...

...

...

...
```

Figure 56: A sample from the content of an *.trak file.

The t.groupTrakTable file contains the tracking information for the groups. Each line lists a correspondence in the file. Similar to the .trakTable file, the group IDs that are on the left side of the deliminators “-1” correspond to the groups from the previous timestep, and the groupIDs that are listed on the right side of “-1” correspond to the groups from the current timestep.

The current TTPN implementation creates only one text-based output file. The name of this file is *Event.list*. Each line in this file represents one object (token) and its history. In each line, the tuple {ObjectID, TimeStep, PlaceID} is written for each timesteps along the movement of the token from an initial place to a final place. However, the recent versions of this file includes only the tuples from timesteps when a change is occurred (i.e. the token is fired) from one place

to another to reduce the file size. In between those tuples, the object is assumed to remain in the same place and therefore that information is not saved in the *Event.list* file. For example, assume that a line in an *Event.list* is 2 1 1 6 3 2 5 4 3. This line can be decoded into the tuples $\{2,1,1\}$, $\{6,3,2\}$ and $\{5,4,3\}$ and can be read as object_2 in timestep 1 falls into P_1 , it becomes object_6 in timestep 3 and changes its place to P_2 and finally it becomes object_5 in timestep 4 and falls into the final place P_3 . Notice that the tuple for the timestep2 is missing in this line. This information is already inherently available in the data and can be obtained easily, when the tracking history is combined with the first timestep information. That is the same object remains in P_1 in timestep 2. And its objectID in timestep 2 can be obtained from the tracking history (provided by the tracking). Therefore there is no need to save that information in the *Event.list* file.

APPENDIX – III:

DATA SETS

In this dissertation, we use three different data sets from 3 different domains. These three data sets are described in this appendix.

A3.1. Pseudo-spectral simulation of coherent turbulent vortex structures

The first data set we use in this dissertation is a small data set from [72] which is a pseudo-spectral simulation of coherent turbulent vortex structures. The simulation includes an initial condition of six vortex tubes in parallel and orthogonal positions. To maintain the energy of the low wave number modes constant, a forcing scheme is applied [17], [21], [71]. The output of this simulation at each time step is saved in a separate file. These files (the dataset) contain vorticity values. The simulation data resolution is 128^3 and it contains 100 timesteps. The data type is a uniform grid and saved in binary files.

This data set contains many merging and splitting features. It has become the typical (experimental) data set and used in many feature extraction and tracking related research including the papers including [72], [73] and [50].

A3.2. Acoustic plume scans

Besides using simulation data, in this dissertation, we include a real data set to apply our techniques on. The real data set is formed of acoustic scans of an underwater plume.

In turbulent underwater environments, hydrothermal plumes can be observed. Such underwater plumes are of the interest of many geologists and oceanographers as in [87] and [8]. As these plumes rise, they bend in response to ocean currents [66]. It is currently of the interest to analyze the relation between the underwater plumes' behavior and the ocean currents. Scientists collect data in various ways to analyze such relation. For example, while earlier studies used ship-based scanners to capture data from the plume, the recent studies use more advanced and stationary systems to scan a plume underwater.

The plume data used in this dissertation is a scan of a plume that is located above the clusters of smokers on Grotto mound, Main Endeavour Field, Juan de Fuca Ridge. The volumetric plume data contains increased backscatter intensity from the metallic sulfides and temperature fluctuations within the plume. The intensity values are collected by using an acoustic scanner. As shown in *Figure 57*, the scanner scans a slice of the plume at a time. Each slice is obtained by elevating the scanner. Then these slices are combined and processed to form a complete 3D data set representing the acoustic measurements of the 3D volume.

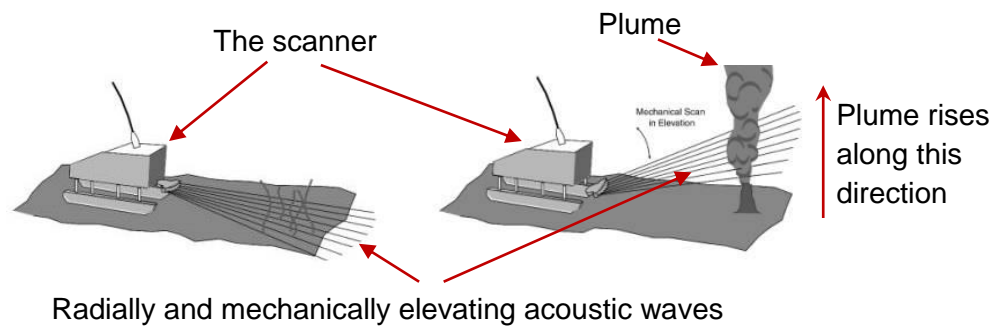


Figure 57: A schematic diagram of acoustic plume scan process. In this figure, the acoustic scanner collects the data by casting out a set of acoustic waves and then by measuring the reflection in the received acoustic waves. The 3D structure is captured by elevating the scans incrementally in short time intervals. Image is modified from [68].

Two different data sets capture the behavior of the plumes from Grotto Vent in the Main Endeavour Field on the Juan de Fuca Ridge. A 24-hr time series (with hourly sampling) data set was collected in 2000 using the ROV Jason. Figure 41a shows one of three existing plumes in the 15 timesteps of the data with 51^3 resolution. The data type is a uniform grid in 3D. A three week time series (with sampling every three hours) was collected using the NEPTUNE Canada cabled seafloor observatory (479 timesteps available) in 2011. As the data set becomes too large to process and view manually, activities of interest need to be modeled and searched for automatically.

A3.3. Wall bounded turbulence flow simulations

In turbulence studies, scientists simulate the environments and conditions to analyze and understand turbulence. Recent focus includes studying and understanding the collective and coherent behaviors of vortices in wall-bounded

flows. Such collective and coherent vortices form groups called packets. In [1], it is proposed that these packets grow in size over time. Moreover, as they evolve, they result in nested packets. These nested packets consist of hairpins or cane-type vortices that are growing up from the wall because the older packets give rise to the younger and slower packets. The term “hairpin” refers both to symmetric horseshoe-like vortices and asymmetric cane-like vortices.

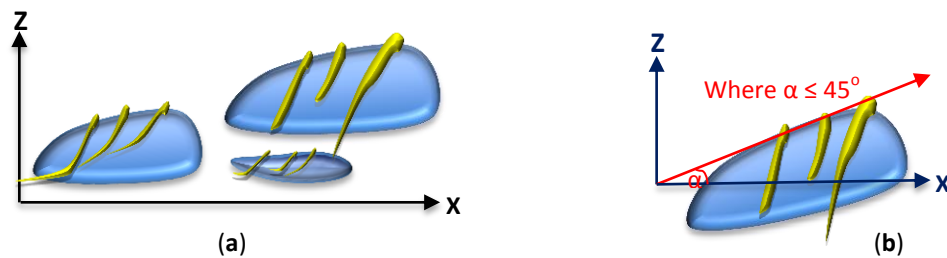


Figure 58: in wall bounded turbulence simulations (a) illustrating the groups of hairpin vortices (packets). (b) An illustration of a packet in wall bounded turbulence simulations. Yellow hairpins elongate to form a certain angle that is smaller than 45° .

The turbulence structures in boundary layers have been studied in various flow regimes including subsonic, supersonic and hypersonic flow regimes examples can be found in [65] and [53]. An illustration of these turbulence structures is shown in Figure 58b. The yellow hairpin vortices (features) move coherently inducing a secondary (blue) fluid mass of low momentum. These coherent structures are the packets. (This is analogous to groups of humans walking coherently in a crowd or to a school of fish). Notice that, in general, the hairpins are not connected within a packet. Each packet includes varying number of hairpins where these hairpins should be aligned at a downstream-leaning angle (γ) and the distance between the hairpins should not exceed a predefined

physically meaningful value (see Figure 58b). Some of these packets lead to the formation of younger packets over time. Moreover, among all these packets, some act coherently forming super structures inducing meandering regions of low momentum. (an illustration of such super structures is given in Figure 58a).

The time-varying wall-bounded turbulence simulation data used in this dissertation contains two different data sets. Both sets contain various variables including temperature, velocity and swirling values for each voxel in each time step. The first data set is saved in a binary format and later on converted into the rectilinear VTK data format. Please refer to [69] for more information about different VTK data formats. The data dimensions are 384x256x69 and the data is contains 46 timesteps.

The second data set is a larger data set and is saved in the rectilinear format by using the hdf5 libraries. Hdf5 is a “file” format and is used by many scientists. It is flexible and supports distributed systems and various programming languages such as C, C++ and Fortran. Please refer to [28] for the details of the HDF5 format. The data dimensions of this data set are 2520x1120x110 and it contains 250 timesteps. The data produced at each time step is saved in a separate file and each file size is about 16 gigabyte.

The activity we are interested in wall-bounded turbulence simulations is the “packet formation” event which is characterized by a single hairpin evolving into a packet formed of multiple features over time.