RWTH AACHEN UNIVERSITY

Event Detection in Videos based on Object Trajectories

Bachelor Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Anne Kathrein

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Bastian Leibe

Registration date: May 30th, 2011
Submission date: Sep 28th, 2011

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

$Aachen, MONTH\,YEAR$
YOUR NAME

Contents

List of Figures

List of Tables

Abstract

This thesis presents an event detection software to accelerate search tasks on videos. In areas like behavioral research, sports analysis, visual surveillance, ethnography and video editing a lot of labor is investigated in video browsing and reviewing. To shorten search time on the video data an event detection system is presented, which enables users to select objects and define search criteria on them. The system returns all video sequences, where search criteria are fulfilled.
Most presented approaches on event detection are fixed to a specific application area and afford a large setup. To avoid this specificity, search patterns were clustered from important events occurring in the application areas. Clustering was performed with respect to shape and constellations of the according object trajectories. The provided system implements seventeen recognition algorithms for the defined patterns. Detection algorithms and their precision-recall-values are presented.
To evaluate the user-interface and acceleration of search time a user study was designed, in which users were provided with search tasks on videos. Assignments had to be completed on the event detection system and on a typical timeline-slider system. The evaluation of the results shows that task-completion was performed significantly faster using the event detection software.

Überblick

Diese Arbeit präsentiert eine Ereignis-Erkennungs-Software, welche Such-Aufgaben in Videos beschleunigen soll. In den Feldern Verhaltensforschung, Sportanalyse, Videoüberwachung, Ethnographie und Videoschnitt wird ein großer Anteil an Arbeit auf das Durchsuchen und Nachsuchen von Videomaterial verwendet. Um die Suchzeit auf Videodaten zu kürzen wird ein Ereignis-Erkennungs-System präsentiert, welches den Nutzern ermöglicht Objekte innerhalb des Videos auszuwählen und Suchkriterien darauf zu definieren. Das System liefert dann alle Video Sequenzen, in denen die Suchkriterien erfüllt sind, zurück.
Die meisten bereits entwickelten Ansätze zu Ereignis-Erkennung sind auf ein spezifisches Anwendungsgebiet fixiert und setzen ein relativ großes Setup voraus. Um diese Spezifität zu vermeiden, wurden wichtige Ereignisse aus den jeweiligen Anwendungsfeldern zu Suchmustern gruppiert. Das Clustern wurde bezüglich den Formen und Konstellationen der berechneten Objekt-Trajektorien vorgenommen. Die Implementierung des dargebotenen Systems enthält siebzehn Erkennungs-Algorithmen für die definierten Muster. Erkennungs-Algorithmen und die entsprechende Exaktheit der Algorithmen werden präsentiert.
Um die Benutzer-Schnittstelle und die Beschleunigung der Suchzeit zu bewerten, wurde eine Nutzer-Studie entworfen, in der Nutzer Suchaufgaben auf Videos erhielten. Diese Aufgaben mussten auf der Ereignis-Erkennungs-Software, sowie auf einem regulären Zeitleisten-Slider absolviert werden. Die Evaluierung der Ergebnisse weist auf, dass Aufgaben mit Ereignis-Erkennung signifikant schneller gelöst wurden, als mit dem Slider-System.

Acknowledgements

First of all, I would like to thank Prof. Dr. Borchers who first brought me in touch and was able to fascinate me for the field of HCI and multimedia. I am very grateful for having the chance to write my bachelor thesis under his supervision. Furthermore, I would like to express my gratitude to Prof. Dr. Leibe for the second revision of my thesis. A special thanks goes to Thorsten Karrer and Moritz Wittenhagen, who took me as bachelor thesis student and offered me the chance to work on the direct object manipulation software *DRAGON*. I want to thank them for the support and advice they gave me during the last six months. I would also like to thank Benjamin Denning, research manager of the GIM, and Benjamin Zipser, behavioral researcher from the Department of Behavioural Biology of the University of Muenster for providing me with information on their fields. I thank everyone who participated in my user study and enabled me to evaluate my system. Especially, I thank my family and everybody else who supported me in the last few months. Thank you!

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of algorithms are set off in coloured boxes.

ALOGRITHM:
HELLOWORLD()

Definition: *Alogrithm*

Parameters of pseudo codes are written in italic text.

parameter

The whole thesis is written in American English.

Chapter 1

Introduction

Videos are well known as a medium for information sharing, entertainment and capturing special moments as in vacation videos. Apart from the average usage a whole sector exists, where video data is utilized and processed for research, commercial use and surveillance.

Behavioral Research

In behavioral research scientists observe the reactions and behavior of animals in test situations. Scientists film experimental-setups to be able to review the test for analysis. The video material is beneficial, since the experiment must only be surveilled by one person while important scenes during the test can be reviewed innumerable times.

Video Ethnography

Video ethnography enfolds a field where analysts film people in different situations surrounded by their natural environment. Important is that the subject does not feel observed, thus the researcher is mostly not present during the video tapings. Examples where video ethnography is performed are market research (GIM) or medical research (Babic [2010]), where the analysts want to find out how consumers interact with the inspected products. Without the produced video material the observers would not be able to grasp people in their natural environment.

Sports Analytics

Sports analytics is a sector, where video analysis is performed on both a private and a commercial level. Players, coaches and clubs review matches to improve their tactics, while companies like the Sports Analytics GmbH (Spo) perform immediate and professional sports analysis for commercial

Video Editing

Visual Surveillance and Forensics

use like commentaries or support of professional sports teams. The main goal in the field of video processing and editing is to select and discard scenes and pace them to a presentable movie. The editors often collect multiple takes from the same action (Mackay and Davenport [1989]). According to Mackay and Davenport [1989] the editor browses through these multiple takes to select the best composition to achieve a dramatic pacing. In visual surveillance and forensic analysis the analysts collect and observe data from multiple cameras (Larry Huston and Pillai [2004]) to be able to protect people or objects from humiliation or damage. Mostly the surveillance is performed by human beings. In forensic analysis the scientist tries to reconstruct and interpret the events. Because of the high number of cameras this task is rather complex, especially when the forensic scientists are asked to perform real-time guidance while the situation is developing (Larry Huston and Pillai [2004]).

Consumed amount of video data.

Everyday the amount of video data increases since researchers perform experiments regularly, crime occurs around the clock and sports matches are carried out nearly each day. One only has to imagine the number of videos that are uploaded onto platforms like youtube, where the number increased by 200.000 per day, to understand what problems small groups of scientists and analysts face, when producing and reviewing video material. Benjamin Zipser, M. Sc. Biol. from the department of behavioral biology of the university of Münster, states that they have diverse experimental situations, in which they take videos of animals. This data needs to be annotated and stored for analysis, if important situations like two animals interacting or showing aggressive behavior occur. The GIM (Gesellschaft für Innovative Marktforschung) performs market research on the basis of ethnographic video tapings (GIM). They search for influences from the context the people come from on the interaction with the analyzed product and for consumer habits. According to Benjamin Dennig, research manager at GIM, a lot of data is generated in video ethnographic market research projects which needs to be compacted for analysis and presentation of results.

The Phillips Research Group created the *ExperienceLab*. This is a house filled with cameras where multi-disciplinary teams of psychologists, sociologist and designers can observe people's behavior in interaction with products from areas like lifestyle or healthcare, so that researchers get insights into the customers needs (Exp). Another example is a behavioral/ethnographic experiment by Roy et al. In a study on language acquisition, they collected 230.000 hours of video data (Brandon C. Roy and Roy, Roy [2009]). Joachim Gudmundssona [2010] states that an important task in sports analysis is that players, coaches, and clubs analyze football matches, so that with current technology a lot of match data is collected. But also companies like the sports analytics gmbh analyze tennis, soccer, or ice hokey matches. During tournaments they provide their customers with video data in different formats and with immediate analyses (Spo).

Accelerate Search: Current Methods

Since data is not only recorded but often reviewed, multiple times for analysis, the produced amount of data makes long search times in various scenes inevitable. With appropriate search mechanisms the task which analysts and scientists face could be accelerated. Already tracking of objects is used in several of the described application areas. Benjamin Zipser states that one software they use at their department is AnyMaze (Any), which is a rather flexible software where you can track almost any situation as long as only one animal is involved. Already surveillance cameras performing motion tracking are available (Sky). And video annotation software which performs tracking is being developed (Gregor Miller and Ilich [2011]). These are examples how the task of video browsing is yet accelerated. If the users were offered more options for search, finding important scenes in video tapings could probably be performed even faster.

Accelerating Search-Task with Event Detection

In this thesis I will present a software with which users can find predefined events in videos. They are able to select multiple objects, choose one of seventeen search patterns, which are described in detail in chapter 3, and browse through the results instead of searching through the whole video manually. Several approaches on event detection in videos already exist. Some of these approaches are based on predefined scenarios (Gerard Medioni and Bremond [AUGUST 2001], Ahmet Ekin and Mehrotra

[July 2003], Tovinkere and Qian [2001]). They are mostly quite precise in their domain, but afford a large effort in setup. Another group of systems on event detection works on user queries (Chen and Chang [2000]). I will present some of these approaches in chapter 2. In this thesis I will demonstrate an event detection system based on the direct object navigation software *DRAGON* (T. Karrer and Borchers [2008]). *DRAGON* tracks object positions and generates object trajectories, so that the user is able to select objects and items directly by clicking on them in the movie. She then has the possibility to define search criteria on the selected objects. After the system performed the search the user is able to access all frames on which the criteria are fulfilled.

1.1 Thesis Overview

Overview

This thesis holds information on related work, implementation and evaluation of event detection. The next chapter deals with related work. Six different systems are presented that perform event detection in various application areas. In chapter 3 my own work is demonstrated. The functionalities and application areas of the event detection extension are described. The system *DRAGON* created by Karrer et al., which is the basis of the event detection software, is presented in section 3.1. Further on I demonstrate the clustering of relevant events in the respective application areas and describe which situations users can find in videos. For each event description the implementation and how search algorithms are performed is presented in form of pseudo codes. Finally the different components and functionalities of the user interface are demonstrated. Chapter 4 deals with the evaluation of the system. In section 4.1. the performance of the system is presented in form of precision-recall tests. A user-study with 12 subjects, where users perform search tasks on video data, is described in section 4.2. Here test set-up acceleration time and SUS-Questionaire results are presented. Chapter 5 contains a summary and lists some tasks in tracking, user interface and algorithm design that should be accomplished in future.

Chapter 2

Related work

Some approaches on event detection in videos exist, where a previous object tracking step is applied before the event detection is performed. A second group of event detection frameworks are based on the detection of events by recognizing cinematic features and object properties. Event Detection frameworks are often based on a special purpose, e.g. supporting surveillance systems or sports video analysis.

2.1 Automatic Soccer Video Analysis and Summarization

Event Detection based on Cinematic Features

Ahmet Ekin and Mehrotra [July 2003] propose a framework for analysis and summarization of soccer videos. This approach focusses on the detection of events based on cinematic features and object-based features. By exploiting color region detection, shot boundary detection, and shot classification algorithms Ekin et al. are able to detect goal events and actions that involve referees or penalty boxes in real time. Goal detection is performed by assuming that the occurrence of a goal is followed by a pattern of cinematic features. Ekin et al. state that a goal event leads to a break in the game, in which emotions on the field and slow motion replays are shown to the TV audience. Emotions are captured by showing closeups of the players and

(a) (b) (c) (d) (e) (f)

Figure 2.1: The broadcast of a goal: (a) long view of the actual goal play, (b) player close-up, (c) audience, (d) the first replay, (e) the third replay, and (f) long view of the start of the new play. Ahmet Ekin and Mehrotra [July 2003]

the cheering audience. When these cinematic features occur the scene is recognized to involve a goal event. Such a goal event can be seen in figure 2.1. To find interesting referee events Ekin et al. filter out the medium or close up shots. They exploit the distinguishable colored uniforms of the referees to detect his occurrence. Penalty box detection is reduced to the problem of searching three parallel lines. Although this approach can perform summaries of interesting events in real time it is very domain specific. Their framework only functions in the field of soccer, to extend it to other sports new events and new event detection algorithms need to be defined. Ekin et al. focussed on the extraction of very few events in soccer, which does not suffice for the whole analysis of a sports match. The event detection software described in this thesis, will not be able to run in real time as the approach by Ekin et al. does, since tracking needs to be computed beforehand. But since it is not fixed to a specific domain, scenario definition, detection algorithm design, and setup-time are avoided. Furthermore the events which can be found are more flexible even in the field of soccer.

2.2 Detecting Semantic Events in Soccer Games: Towards A Complete Solution

Semantical information in ER-Diagramms

Tovinkere and Qian [2001] present a method for detecting all possible soccer events, which can occur during a match. These events are defined at a semantical level in form of an entity relationship diagram which contains domain specific knowledge. Position information of the players and the ball is required as an input for the event detection algorithm. In a first step tracking data is extracted from the analyzed video. The knowledge domain and the tracking data conduce as input to the event detector. The detection is performed in two phases. First, player motion and orientation and player-ball interaction is computed. In the second step the algorithm determines which rules from the knowledge base are applied to the actions of the players and the ball. With this framework Tovinkere et al. are able to detect events like assist, block, header, interception, shot-on-goal and so forth. Their event detection performance shows convincing precision-recall values. This approach is also very domain specific and applying it to other application areas is elaborate since a whole knowledge base needs to be created previously.

Object Tracking

Comparison to *DRAGON*

The event-detection extension for *DRAGON* can not be as precise in the domain of soccer as the system designed by Tovinkere et al., because this would afford additional domain based knowledge. On the other hand the described approach is so exact in the domain of soccer, that knowledge base creation affords a large setup, thus the system is fixed to a certain domain. *DRAGON*'s event detection extension affords no setup and can be applied in any application area, without changing the system.

2.3 DOTS: Support for Effective Video Surveillance

Event detection for Surveillance Systems

Andreas Girgensohn and Rieffel [2007] describe a real-time multi-camera surveillance system. Events are identified, when objects appear or disappear, when people pass certain regions or when a lot of motion is detected. DOTS detects foreground objects, associates them with a bounding box and tracks these objects. An event is triggered if a lot of motion occurs in a scene, i.e. when the ratio of foreground and background pixels exceeds to a predefined level. Region-based events are triggered when moving objects intersect "hot spots". This way events like opening or closing doors can be detected. The user is able to define hotspots by himself, so that all object trajectories that cross this hotspot are recorded. Furthermore DOTS supports externally generated events produced by sensors or RFID tags. Although the amount of events that can be detected are few, Girgensohn et al. state that the events offer a quick access to the recorded video data.

Hardware Setup

To use DOTS a hardware setup is affordable, which has both benefits and downsides. Additional hardware like RFID tags provide supplemental information that helps interpreting events. A large amount of cameras is used to make localizing people in all rooms possible. But this also means that the event detection software only functions properly on videos recorded with the hardware setup.

Comparison to *DRAGON*

DRAGON does not provide these observational functionalities and has no additional sensor information. On the other hand it does not afford any hardware setup and can be run on arbitrary video data. Furthermore users can select up from 17 search patterns, which also include events like appearing and disappearing.

2.4 Left-Luggage Detection using Bayesian Inference

Left-luggage Detection Framework

Fengjun Lv and Nevatia [2006] present a system for Left-Luggage detection. The left luggage event is based on several constraints: When the luggage enters the scene, it is owned by a person and the physical contact is interrupted

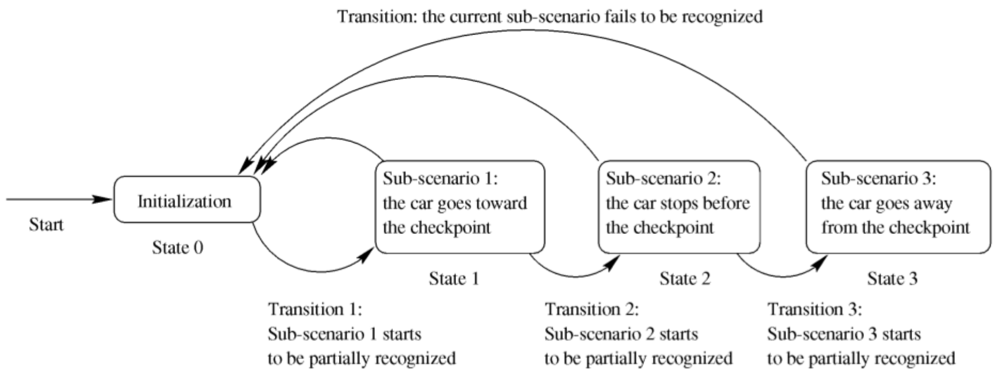

Figure 2.2: Automaton for the Multi-State Scenario a the Car avoids the Checkpoint. [Gerard Medioni and Bremond [AUGUST 2001]]

at some point. The luggage is defined as unattended if the person moves further than 3 meters. An alarm is triggered if the luggage remains unattended for more than 30 seconds. To detect the occurrence of these constraints object detection and object tracking steps need to be performed. The events are then modeled and recognized in a bayesian inference network by assigning probability values to certain events. These events include dropping the luggage, luggage appears or does not appear and the distance of a person to the dropped luggage is less than a certain threshold. By combining the named events the probabilities for the predefined constraints can be computed. This results in convincing event recognition, even if the data is noisy.

Comparison to *DRAGON*

Lv et al.'s approach performs a very precise detection of the left-luggage scenario. The event detection extension of *DRAGON* includes a pattern, with which a constellation where objects are close first, but then move away from each other can be found. This pattern can not detect a left-luggage scenario as exact as the approach of Lv et al. does, but it offers the opportunity to detect this situation next to many other trajectory patterns.

2.5 Event Detection and Analysis from Video Streams

Gerard Medioni and Bremond [AUGUST 2001] present a system, which classifies patterns of moving regions and object trajectories. In a first step their system detects and tracks objects. The user needs to provide information on spatial structures of the scene and expectations on scenarios which might occur. Given the user information and the trajectories, it is now possible to interpret the occurring scenarios. First, each object is classified by some properties like width, height, or location. The behavior of the mobile object is compared against a set of predefined scenarios. After this analysis the system outputs the most likely scenario. Medioni et al. define single state events as a set of sub-scenarios that must be recognized at the same time and multiple state events which are composed of several sub-scenarios occurring in a temporal sequence. Scenarios are classified in form of DFAs, where each node of the DFA contains one sub-scenario and the transitions are computed from recognition values and likelihood degrees of the sub-scenarios. A multi-state DFA is shown in figure **??**. Both single state and multiple state events can be recognized by the finite state automaton. An advantage of this system is that it presents events on a semantical level, but on the downside this affords an elaborate setup of the context information since slight deviations in the context make the system unreliable. *DRAGON* on the other hand contains no semantical information because it is designed for several application areas. Thus a change of the domain has no impact on the performance of the event detection extension for *DRAGON*.

User Input

Event classification via DFAs

Comparison to *DRAGON*

2.6 Motion Trajectory Matching of Video Objects

Trajectory Matching

Chen and Chang [2000] describe an approach for robust motion trajectory matching. Here the user formulates a query by drawing a sketch and assigning areas in the frame. The user can also add color, texture, and shape information. This sketch is then tested against complex trajectories stored in a database. The trajectories are precomputed

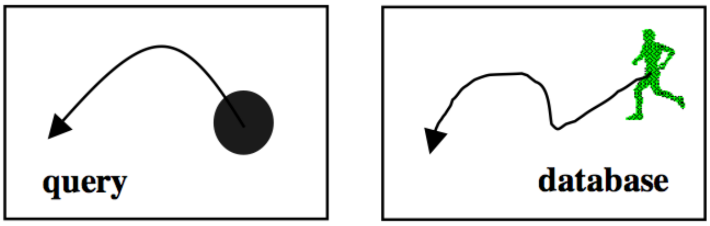

Figure 2.3: Comparison beween trajectory data and input query. [Chen and Chang [2000]]

Smoothing complex Trajectories

by an automatic tracking algorithm. While the objects perform complex movements, Chen et al. state that it is more likely that the user's sketch describes the motion fairly simple. An example can be seen in figure 2.3. To bridge this gap, three steps, smoothing, segmentation, and modeling are applied. Since the trajectories are often noisy they are smoothed in a first step by a wavelet based approach. Then the trajectory is segmented into sub-trajectories with constant acceleration. Finally the sub-trajectories are modeled as feature vectors of acceleration, velocity, arc-length, order, and multi-scale edge-points. The object based search is then performed by measuring the distance between the feature vectors of the trajectories and those of the query. A list of possible candidates is returned. Their approach concludes in convincing results in precision-recall values compared to related methods like e.g. using B-splines for trajectory matching. An advantage of this system is that the user can post clear search queries, but this also means that she must know the characteristics of the object-trajectory she is looking for. In many situations the trajectory-course is unknown to the user and thus she is not able to put a query. The trajectory-shape based algorithms of the event detection extension of *DRAGON* are designed relatively loose, and the distance based algorithms do not assimilate shape information. Hence the user does not need to know the trajectory course beforehand.

Feature Vector

Comparison to *DRAGON*

Chapter 3

Event Detection in Videos based on Object Trajectories

This thesis deals with the development of a software to accelerate search tasks in videos by detecting one of seventeen predefined events. In contrast to most of the approaches presented in related work, this event detection system can be applied to arbitrary video data without affording an elaborate setup, since a broad number of application areas exists. The event detection is performed in two steps. In the first step object trajectories are generated from the video material. This object tracking step is accomplished by *DRAGON*, a direct object manipulation software, which I will present in section 3.1. Second the actual event detection takes place, where tracking data is evaluated and recognized events are extracted and displayed to the user. In chapter 1 I described diverse application areas, where people spend a lot of time watching videos. After clustering relevant events from these application areas to trajectory patterns, I was able to define search criteria on the tracking data. The defined patterns include area-, directional- and velocity dependancies, object motion and object-object interaction. The trajectory patterns are described in detail in section 3.2. To perform search tasks the user can pick one of seventeen patterns. Then she is able to select the objects which she wants to involve in the search task over the *DRAGON*-Interface by directly clicking on them. After the search is performed the relevant frames are marked beneath the timeline, so that the events can be accessed easily. The interface is described at length in section 3.3. To test the event detection software in performance and usability I accomplished precision recall tests and performed a user study with twelve subjects.

3.1 DRAGON: Object Tracking

DRAGable Object Navigation

DRAGON is an in scene navigation software presented by T. Karrer and Borchers [2008]. The user can navigate through a scene by clicking on an object and dragging it along its object trajectory. Figure 3.1. shows the *DRAGON*-Interface. The red line represents the object trajectory along which the user can drag the selected object and by this scroll through time.

Optical Flow

The trajectory generation is based on an approach by T. Brox and Weickert [2004]. In a precomputation step optical flow fields are generated. Flow-fields contain information on the most likely pixel locations in the succeeding and preceding frames. The white arrows in figure 3.1 are examples for flow fields in forward direction. At runtime the precomputed flow fields are used to compute the object trajectory of the pixel the user clicks on (T. Karrer and Borchers [2008]). The computed trajectories contain position information of the defined pixel for each frame. To drag the object along its trajectory Karrer et al. look for the frame with minimal (x,y,t)-distance to the current mouse-pointer location. This means they calculate the closest position in space and time.

Users

In user studies Karrer et al. show that participants found the use of *DRAGON* very natural and that in-scene navigation was performed faster using *DRAGON*. In this thesis I will exploit the trajectory generation *DRAGON* performs and extend the *DRAGON* interface by allowing the selection of multiple objects, on which the search tasks can be defined. Navigating through the found events can be performed in three manners. First one can step through the events by clicking a "next-" or "previous event" button. Second the user can exploit the in scene navigation tools offered by *DRAGON* and finally the user can navigate via the conventional slider. Karrer et al.

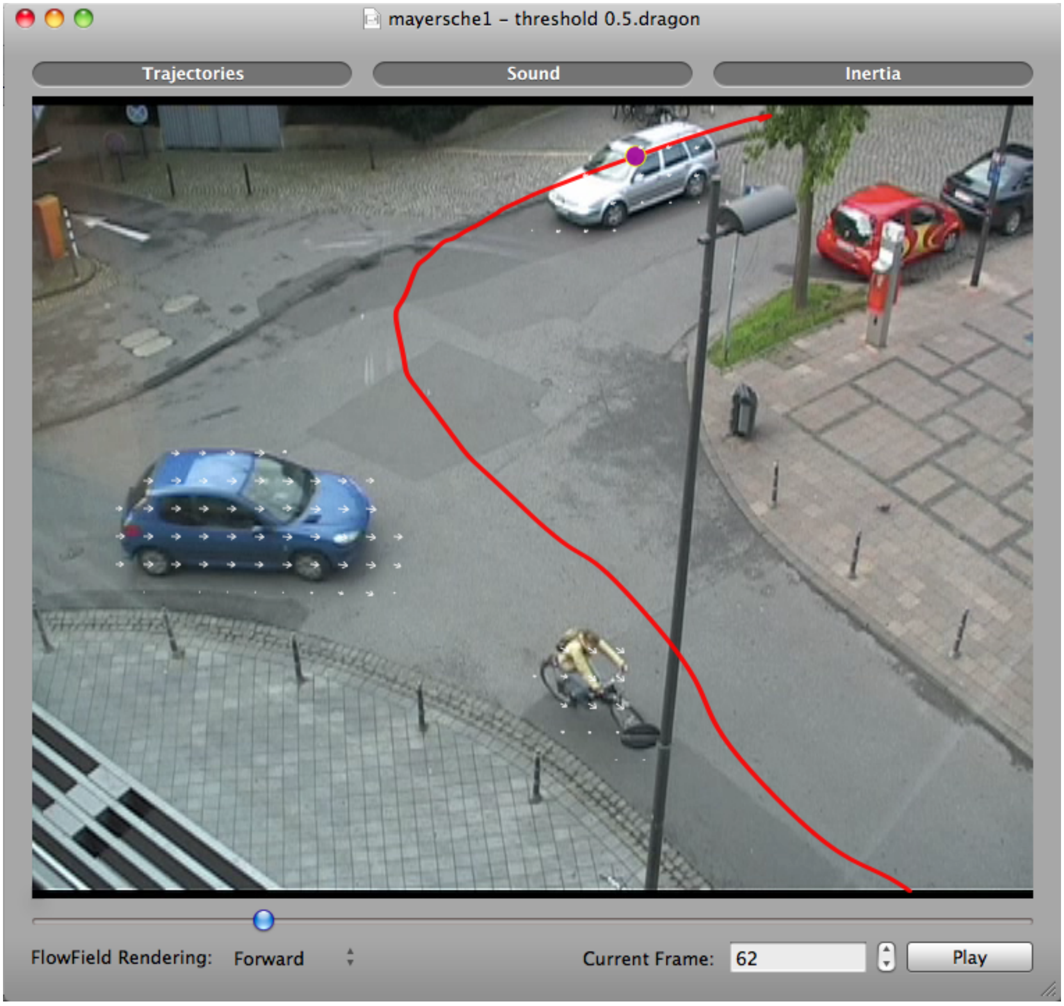

Figure 3.1: *DRAGON*-Interface

state that the participants of their user study on *DRAGON* where interested in having multiple navigation techniques to be able to navigate with alternating accuracies.

3.1.1 Limitations

Disadvantages of Optical Flow

Optical Flow based tracking performs the tracking of single pixels. This results in a lack of object awareness (Wittenhagen [2008]). Since pixels are not clustered to objects, occlusions of tracked pixels can not be handled. This also results in the fact that objects can not be tracked beyond

Camera Motion

a scene cut. By using optical flow fields for tracking the position of the object in the frame can be determined, but it contains no information on the position of the object in the scene. An approach by Wittenhagen exists where *DRAGON* is extended by camera motion estimation. For simplicity this work is limited to video material from still cameras.

3.2 Object Trajectory Patterns

Relevant Events in the Application Areas

In a first step I gathered events from the application areas. Typical events from the area of visual surveillance are intrusion detection (Weiming Hu and Maybankt [12 Juli 2004], Mac), crowd surveillance (Shobhit Saxena and Ma [2008]), which includes deviating velocities or movement direction of single people in crowds, traffic observation (collision detection, speed control (Saunier and Sayed [2007], Gerard Medioni and Bremond [AUGUST 2001])), or lost luggage detection (Fengjun Lv and Nevatia [2006]). In the area of behavioral research important events are animal-animal and animal-object interactions, when animals leave certain areas or when they show aggressive behavior. In market research any interaction with observed products and human habits like cooking, cleaning, watching TV, or shopping are relevant (GIM, Babic [2010]). Interesting sports events include kick-offs, goals, passes, corner kicks, fouls and so forth (Tovinkere and Qian [2001]). By clustering relevant events from the application areas forensic analysis, behavioral research, video editing, ethnography, and sports analytics I defined seventeen patterns, which the users can select. These patterns are grouped into the four main clusters:

Clustering of Trajectory Patterns

1. Area Dependancies
2. Objects Act
3. Objects Interact
4. Direction and Velocity

Four Main Clusters

Each of these clusters is described in detail in the following sections.

3.2.1 Requirements

Trajectory Properties

Figure 3.1 shows the *DRAGON*-Interface. The red line represents the object trajectory along which the user can drag the selected object and by this scroll through time. The

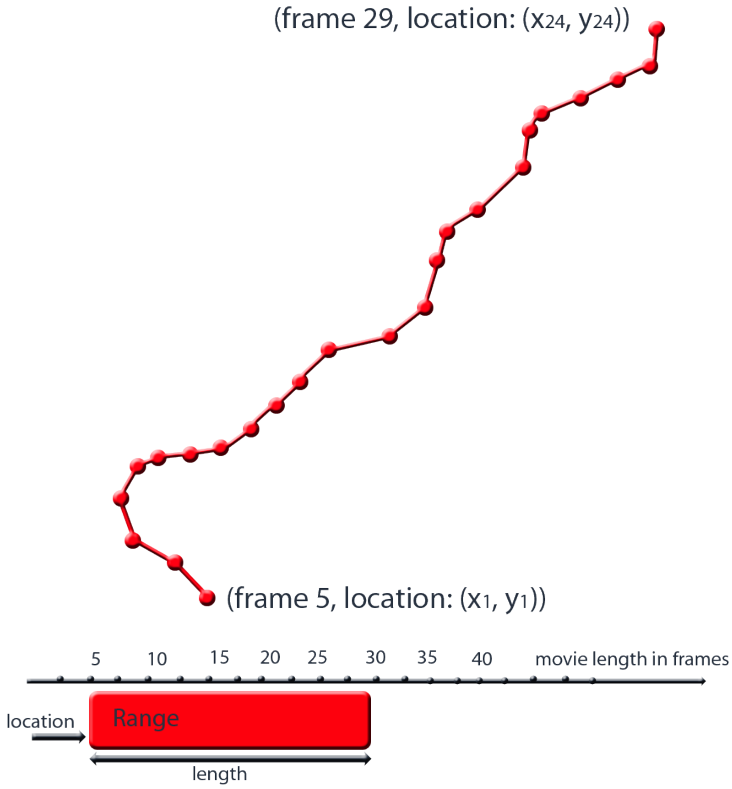

Figure 3.2: Example of an object-trajectory, its properties and its range.

object trajectories, which are computed by *DRAGON* contain one *trajectory vertex* per frame. These vertices consist of a frame number and the position of the object in the respective frame given in (x,y)-coordinates. Trajectory properties are depicted in figure 3.2. In the pseudocodes, describing the pattern recognition algorithms, the access of the trajectory properties are denoted with *trajectory.node.position* and *trajectory.node.frame*. By calling the function STOREDFRAMES(Trajectory *trajectory*) on a trajectory it is possible to gain the *frame range* on which the trajectory is defined. The returned range contains a *location*, denoting the first frame and a *length* value describing the number of frames, where the object appears in the video.

Areas

Areas are defined in form of rectangles, which contain an offset point, width, and height.

Dependancies

If more than one trajectory is involved in the search, the user can define a *dependence variable*, which can only take the values T_1 or T_2. This dependance value can have different meanings, which will be explained in every pattern description separately.

Functions

Several functions, used in the pseudo codes, have rather simple implementations and are not explained in more detail:

- FRAMESFROMRANGE(Range *range*): Returns a set containing all the frames in range *range*.
- POSITIONFORFRAME(Trajectory *trajectory*,int *frameNr*): Returns the location of the trajectory *trajectory* at *frameNr* in (x,y)-coordinates.
- STOREDFRAMES(Trajectory *trajectory*): Returns the the frame-range in which the *trajectory* is defined.
- GETFIRSTFRAME(Trajectory *trajectory*): Returns the first frame where *trajectory* is defined.
- GETLASTFRAME(Trajectory *trajectory*): Returns the last frame where *trajectory* is defined.
- INITWITHCAPACITY(Integer *capacity*): Returns an array with size *capacity*.
- SIZEOF(Array *a*): Returns the number of entries in *a*.
- SETWITHOBJECT(Object *object*): Returns a set containing *object*.

- ARRAYWITHOBJECTS(...): Returns an array containing the objects from the input.
- INTERSECTRANGE(Trajectory t_A, t_B): Returns the frame range where both trajectories are defined.
- POINTSINRANGE(Trajectory *t*, Range *range*): Returns an array, which contains the location points of *t* in frame range *range*.
- UNIONRANGE(Array *trajectories*): Returns the smallest coherent range where all ranges of the trajectories from the input array are defined.

3.2.2 Area Dependancies

Cluster: Area-Object Interaction

The interaction of objects with areas is an important and often searched event. An area can describe a special location in the background or a still object which needs a lot of space, e.g. a soccer goal. Area-events are searched in many application areas. For example Benjamin Zipser states that he often analyzes situations, in which an animal is brought into a test situation, where he wants to find out how long it takes until an animal leaves the safe area of the test box, or how long an animal explores an unknown object. Joachim Gudmundssona [2010] describes that corner kicks and kick offs are interesting events and Ahmet Ekin and Mehrotra [July 2003] sees goals as one of the main events in soccer. Weiming Hu and Maybankt [12 Juli 2004] and Mac describe intrusion detection and access control into special areas like important government units or military bases as relevant events in visual surveillance. Also, Benjamin Dennig from the GIM states that one example for an everyday situation is cooking. Events like loading or looking into the oven are important interactions, which need to be observed. Each interaction close to the area of the oven is important. The described events can be found by defining an area and searching for events where an object is located inside of the defined area or when it avoids the area. In this group of patterns the user can define an area in form of a rectangle and search for the patterns *Object Crosses Area* and *Object is Far from Area*. An example is depicted in figure 3.3.

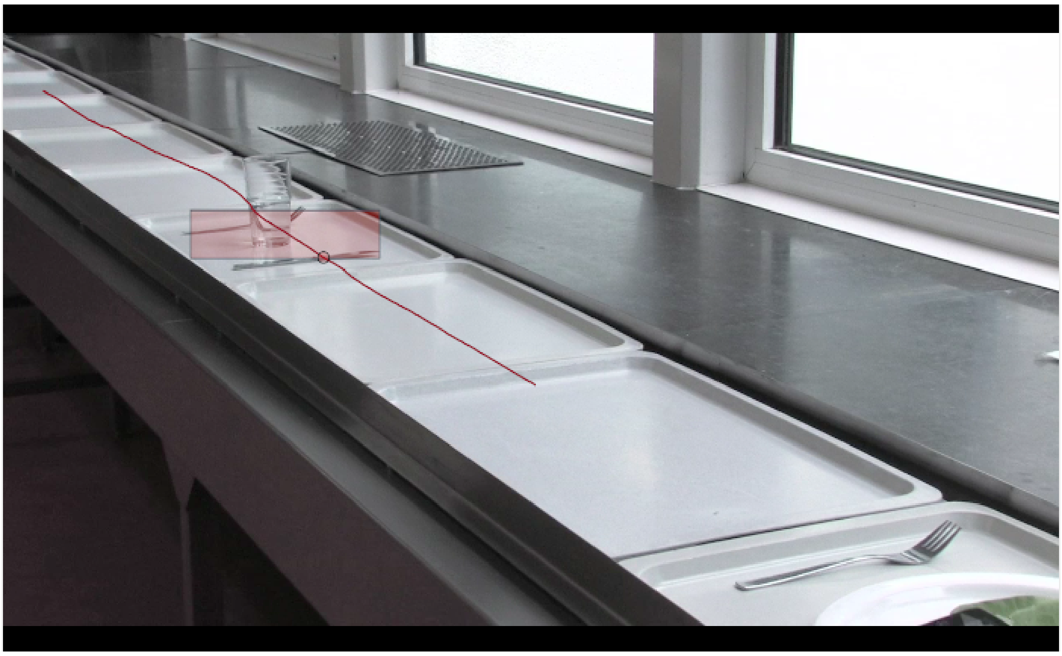

Figure 3.3: Object trajectory of the knife with a selected area. (Video by Karrer et al.)

Object Crosses Area

Pattern-Implementation: Object Crosses Area

With the pattern *Object Crosses Area* the user is able to find frames, where the defined objects are inside of a selected area. The algorithm 3.2.1 returns the set of frames where a given trajectory crosses a user defined area by intersecting the *area* pixel-positions (GETPIXELSAREA(Area *area*)) with those of the *trajectory*-nodes. Algorithm 3.2.2. gets an area, an array of trajectories, and a dependance value as input. For each trajectory the function TRAJECTORYCROSSESAREA() is called. If the dependance value is equal to T_1 the trajectories need to be in the area at the same time and the algorithm only returns those frames where all defined trajectories are inside of the area. Otherwise, if it is not necessary that the trajectories cross the area at the same time, the algorithm returns all frames where a trajectory-node location is inside of the area.

OBJECT CROSSES AREA :
Algorithm 3.2.1: TRAJECTORYCROSSESAREA(area, trajectory)

local Set setOfFrames $\leftarrow$ {trajectory.node.frameNumber||GETPIXELSAREA(area)$\cap$ trajectory.node.position| ≥ 1}
return (setOfFrames)

OBJECTS CROSS AREA:
Algorithm 3.2.2: TRAJECTORIESCROSSAREA(area, trajectories, dependance)

local Array arrayFramesInArea $\leftarrow$ NULL
local Set setOfFrames $\leftarrow \emptyset$

for $i \leftarrow 0$ **to** SIZEOF(trajectoryArray)
  do arrayFramesInArea[i] $=$ TRAJECTORYCROSSESAREA(trajectoryArray[i],area)
if (dependance $== T_1$)
  then
  for $i \leftarrow 0$ **to** SIZEOF(arrayFramesInArea)
    do setOfFrames = setOfFrames $\cap$ arrayFramesInArea[i]

  else if (dependance $== T_2$)
  then
  for $i \leftarrow 0$ **to** SIZEOF(arrayFramesInArea)
    do setOfFrames = setOfFrames $\cup$ arrayFramesInArea[i]

return (setOfFrames)

Object is far away from Area

Pattern-Implementation: Object is far away from Area

The pattern *Object is far away from Area* finds frames where objects are located outside of the selected area. Algorithm 3.2.3. takes as input a trajectory, an area, and a distance value, which the user defines. It returns a set of frames where the trajectory is not within distance *distance* from the *area*. For each frame where the trajectory is located outside of the area the algorithm checks the minimum distance between the area and the current position of the trajectory. The distance is computed by the function MINDISTANCETOAREA(), which performs an orthogonal projection of the trajectory position onto the edges of the area-bounding box. This process is visualized in figure

Distance Computation

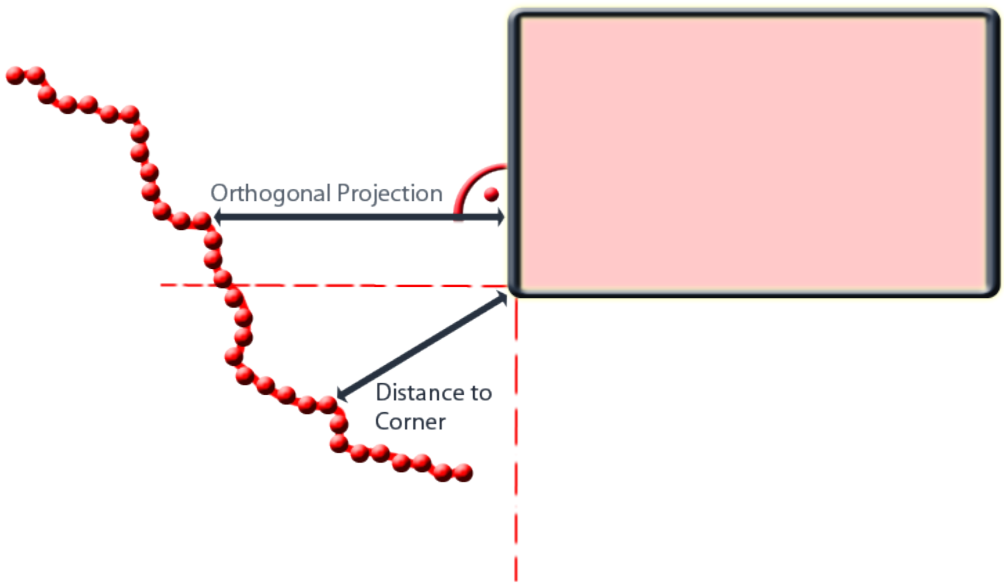

Figure 3.4: Distance Computations with Areas: Shows distance analysis of a trajectory (left) with an area (right). Inside of area boundaries the orthogonal projection is computed, outside of area boundaries the distance to the closest edge-point is calculated.

3.4. If this distance is less or equal to the input distance the frame is removed from the set of resulting frames. Analog to the algorithm 3.2.2 an extension for multiple trajectories is computed in the same manner. The user can also provide a dependance value indicating if the trajectories need to be far away from the area at the same time.

OBJECT FAR AWAY FROM AREA :
Algorithm 3.2.3: TRAJECTORYFARFROMAREA(area, trajectory, distance)

```
local Set setOfFrames ← FRAMESFROMRANGE(STOREDFRAMES(trajectory))
setOfFrames ← setOfFrames\TRAJECTORYCROSSESAREA(trajectory, area)
for each frameNumber in setOfFrames
  do
  { local point ← POSITIONFORFRAME(trajectory, frameNumber)
  { if MINDISTANCETOAREA(area, point) ≤ distance
  {   then REMOVEFROMSET(setOfFrames, frameNr)

return (setOfFrames)
```

3.2.3 Objects Act

Cluster: Objects Act

The *Objects Act* cluster contains trajectory patterns, which are not influenced by other trajectories, so that each trajectory can be analyzed separately. This cluster contains the patterns *Objects Appear*, *Objects Disappear* and *Object-Trajectory forms a Circle*. The first two patterns are one of the main events described in the DOTS surveillance system (Andreas Girgensohn and Rieffel [2007]). Also, a software for object tracking and notes insertion in videos developed by Favalli et al. contains a special feature that alerts the user when an object disappears [Lorenzo Favalli and Moschetti [APRIL 2000]]. The behavioral researcher Benjamin Zipser states that aggressive behavior of animals contains relevant events for their research. According to the Denver Municipal Code this includes animals encircling their victim (Den [1950]). Furthermore a study on Gobiid Fish shows that the fish ”circle” each other to attack their opponent (Yanagisawa [1982]). But also situations where a person returns to a location, where he has been before, e.g. when she is walking around an object of interest, can be found.

Objects Appear and Objects Disappear

Pattern-Implementation: Objects appear and disappear

The algorithms 3.2.4 and 3.2.5 get as input a trajectory. To find out when the object appears or disappears the system returns the first and respectively the last frame where the trajectory is defined. The function is called on all defined trajectories.

APPEARANCE AND DISAPPEARANCE :

Algorithm 3.2.4: OBJECTAPPEARS(Trajectory trajectory)

return (GETFIRSTFRAME(trajectory))

Algorithm 3.2.5: OBJECTDISAPPEARS(Trajectory trajectory)

return (GETLASTFRAME(trajectory))

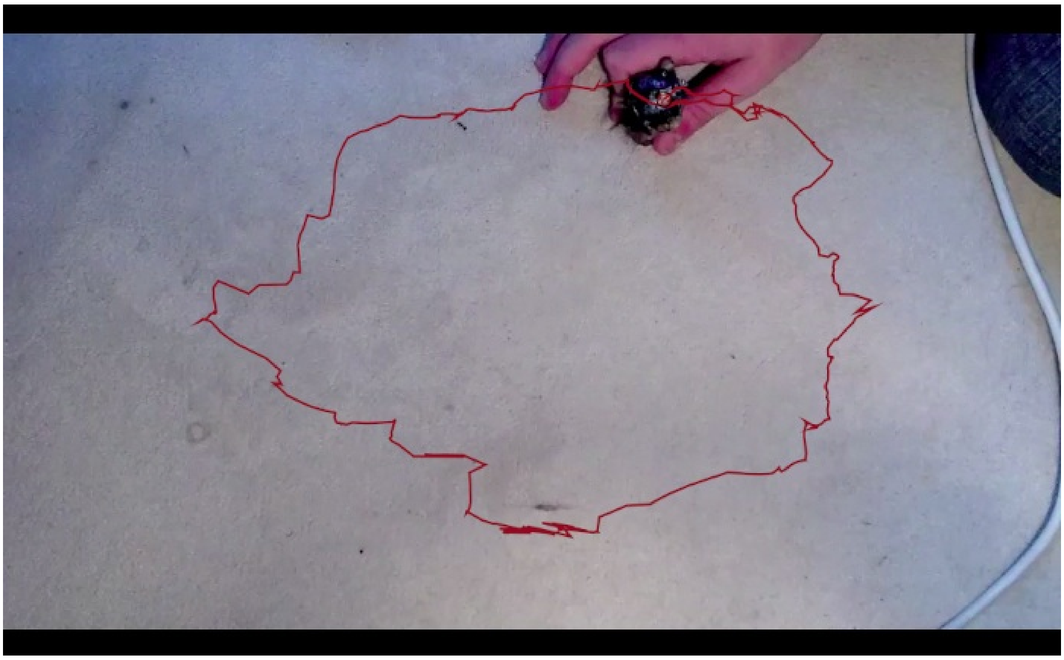

Figure 3.5: Object trajectory of a troll figure performing a circular motion.

Object-Trajectory forms a Circle

Pattern-Implementation: Circle

With the pattern *Object-Trajectory forms a Circle* the users are able to find situations where object-trajectories form circle-like structures as seen in figure 3.5. It is not important that the object moves in a perfect circle, which is why there is a relatively loose variability in the threshold values. I analyze the points of the trajectory with respect to six constraints, that a circular shaped polygon should satisfy. The recognition of the pattern *Object-Trajectory forms a Circle* consists of the following seven steps.

Identifying a Circle

Self-Intersections

1. Find intersection point: A trajectory only forms a circle if the object returns to a position where it was located before, so in a first step all self-intersections of the trajectory are computed.

Remove Sub-Circles

2. Remove Sub-Circles: If the trajectory holds sub-circles (e.g. like trajectories that forms an 8), the sub-circles are removed and examined separately. The points that remain from the first two steps represent the input for algorithm 3.2.6.

OBJECT-TRAJECTORY FORMS A CIRCLE:

Algorithm 3.2.6: ISCIRCLE(Array circlePoints)

if not CHECKBOUNDINGBOXSIZE(threshold)
  then return (**false**)
if not CHECKBOUNDINGRATIO()
  then return (**false**)
if not CHECKAREA()
  then return (**false**)
if not CHECKDISTANCEHISTOGRAMM()
  then return (**false**)
return (**true**)

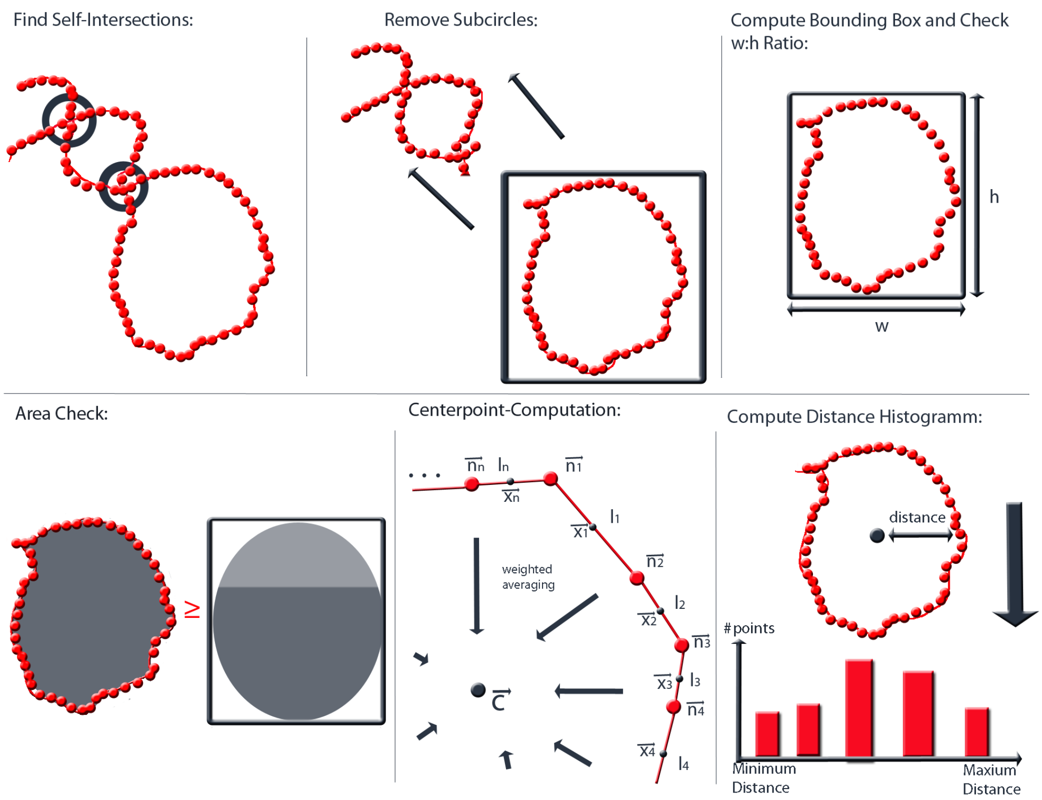

Figure 3.6: Visualization of Circle-Detection showing object trajectories (red) with nodes n_i, bounding boxes (blue), and a distance histogramm. x_i denote the center points and l_i the length of intermediate lines.

3. Check Bounding Box Size: A bounding box is created around the trajectory. If this bounding box is smaller

Bounding Box

than a certain user defined threshold, the trajectory nodes are not recognized as a circle.

w:h ratio

4. Check Bounding Box w:h Ratio: If the bounding box does not show a rate less that 2:1 and greater than 0.5:1 in width:height the "circle" is to flat and not recognized as a circle anymore.

Area Check

5. Area Check: Checks if the trajectory area A encloses at least 60% of an area which a perfect ellipsoid located inside of the bounding box would hold. The percentage 60 has been tested on several videos, which showed satisfying results.

$$A \geq 0.6 \cdot \pi \cdot \frac{\text{width}}{2} \cdot \frac{\text{height}}{2} \tag{3.1}$$

The area is computed with the **marching corner cutting** algorithm (Sandip Sar-Dessai and Kobbelt). In each step of the algorithm a convex corner (or respectively a triangle) is cut off of a polygon, if there is no other point contained in this triangle. This polygon is constructed from the points of the trajectory that form the circle in the order of their appearance. When only three points are left the triangle surfaces are added.

Center Point

6. Compute Center Point $c = (x_c, y_c)$: In a first step the center points x_i and the lengths l_i of the N edges of the circle polygon are computed. From these points the center point is computed by averaging the x and y values for compound linear balance point computation (Cen).

$$x_c = \frac{1}{L} \sum_{i=0}^{N} x_i \cdot l_i \tag{3.2}$$

$$y_c = \frac{1}{L} \sum_{i=0}^{N} y_i \cdot l_i \tag{3.3}$$

$$L = \sum_{i=0}^{N} l_i \tag{3.4}$$

This way all polygon nodes influence the center point computation relative to the weight of their neighboring edges.

7. Compute Distance Histogram to Center Point: A distance histogram containing the distances from each point to the center point is computed. If at least 50% of the points should lie in a similar distance from the center point the histogram check is succeeded.

Distance Historgramm

For explanatory the single steps are visualized in figure 3.6. Algorithm 3.2.6. gives an overview of the described steps. It returns true if the points in the input array *circlePoints* form a circle-like geometry. Here the first two steps are already accomplished. The functions which are called in this algorithm perform the checks described above.

3.2.4 Objects Interact

Cluster: Objects Interact

The interaction of objects include meaningful situations in many application areas, which is why *Objects Interact* is the most extensive cluster. Most patterns in this group analyze the trajectories based on their distance to each other. In sports, events like fouls or backing, where two people are close to each other, or when the ball is caught in the goal are interesting events (Tovinkere and Qian [2001]). According to Benjamin Zipser, behavioral researchers find that all interactions of animals, e.g. when they are close, far away, run away from each other, or follow each other are relevant. Analysis examples are how long animals stay together or remain far away from each other. Benjamin Dennig states that every interaction with an analyzed product is important in ethnographic market research. Also, Babic [2010] states that the interaction of the patients with medical products is relevant for their analysis. In visual surveillance and forensics a person bumping into someone else or showing aggressive behavior is evaluated as suspicious or alerting behavior. There are several forensic studies on aggressive behavior of people. For example, in a study by D'Orio et al. on the reduction of seclusion and restraint in a psychiatric emergency service, the researchers found out that seclusion occurs due to ineffective management of problematic situations. Violent and aggressive behavior need to be recognized fast and escalating situations need to be managed, which is why video surveillance was increased (Ori [2004 American Psychiatric Association]). Furthermore collision detection in traffic surveillance is an important event (Saunier and Sayed [2007]).

Distance Patterns

To detect these scenes I defined several distance patterns. With the patterns *Distance between Objects: Small*, *Distance between Objects: Big*, and *Distance between Objects: Increases*, the user can find frames where objects are close, far away, or when two objects move away from each other. She has the probability to scale the minimum and maximum distances. The pattern *Objects meets* additionally allows to find frames where an object-trajectory is within intersection distance of a different trajectory, independent of time. This is especially useful if the user wants to search for scenes where one object follows another object, or when e.g. market researchers want to find points of interest, which are observed by many people after an other. The pattern *Object meets several other Objects* allows to define a special object. Only *its* object trajectory is checked for intersection points with the other defined object trajectories. This kind of pattern is useful in settings as the Phillips Research's ExperienceLab [Exp], where the user wants to find out which of the objects have been observed by a person. The intersection of all defined object trajectories in one point would not be very useful in this case. The pattern *Objects are visible in the same Frame* enables the user to find frames, where interactions of the defined objects are possible at all. It only finds those frames where all objects are visible.

Left-Luggage-Detection

An other important crime pattern, that has also been content of the Pets workshop 2006, is the left-luggage scenario. Here the person carrying the luggage is close to the object first and deviates at a certain point, while the luggage stays still (Mac, Fengjun Lv and Nevatia [2006]), which is why I defined the pattern *Distance between two Objects increases after a close Motion*.

Crowd Surveillance

An other interesting topic in forensic analysis is crowd surveillance. In [Shobhit Saxena and Ma [2008]] the crowd motion and behavior is analyzed to e.g. be able to perform crowd management. Therefore the similarity of the crowd motion is computed. A pattern to find similar motion of objects is *Objects have Parallel Object Trajectories*, which checks the similarity of the geometry of the trajectory-lines.

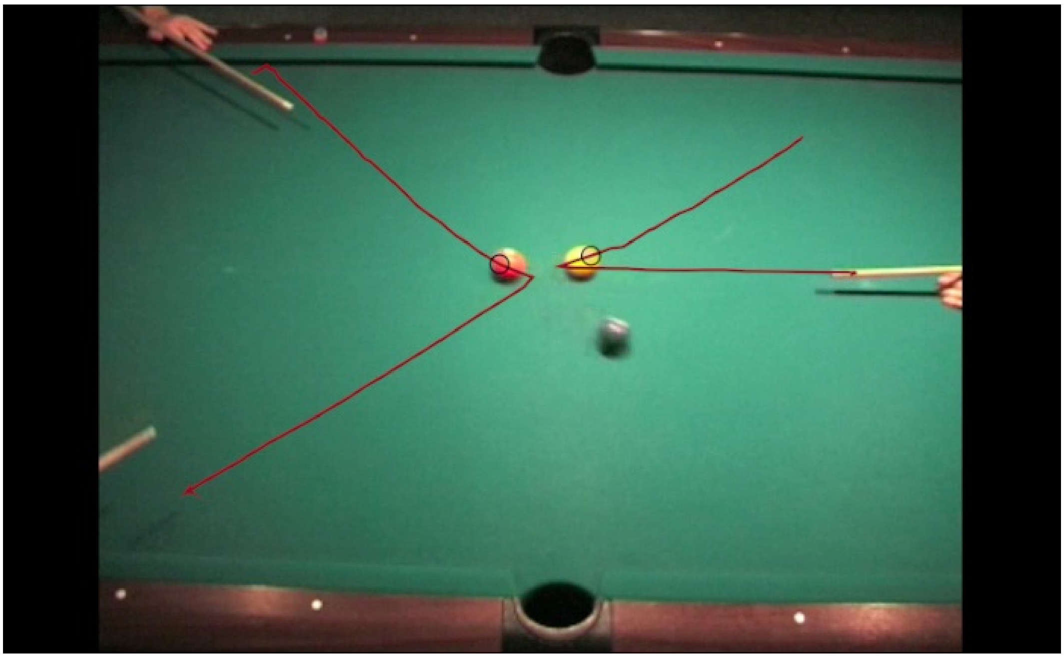

Figure 3.7: Object trajectories of billiard-balls , being far away, then close when they hit each other and then the distance increases. Each situation is recognized by the respective algorithm. (Video by Karrer et al.)

Object Passing

Important events in sports are passes (Tovinkere and Qian [2001]). Here the object *ball* is at first close to a person. After the pass it deviates from this person and arrives at a different player. A similar pattern can be observed in visual surveillance. One contribution in [Mac] is the analysis of trajectories by detecting suspicious object transferring in order to detect robbery situations. The object transferring occurs in a similar manner, only with a smaller distance between the involved objects. To detect these situations I defined the pattern *An Object Moves from one Object to an Other*.

Distance between Objects: Small

Pattern-Implementation: Distance between Objects: Small

With the pattern *Distance between Objects: Small*, the user is able to find frames where objects are close to each other. The implementation of this pattern is described by Algorithm 3.2.7. It returns all frames of a video, in which the distance of all objects is less than or equal to *maxDistance*. The value *maxDistance* is selected by the user. In a first step the set of resulting frames is initialized by all frames in which the first object-trajectory is defined. Then the object-location is checked against the other trajectories for each frame. As soon as one of the trajectories is not within *maxDistance* to one of the other trajectories in a frame or when a trajectory is not defined on a frame this frame is removed. The first constraint is checked by computing the magnitude of the vector *pointA-pointB*. Additionally it needs to be checked if the neighboring lines of the points intersect. This is done due to the fact that trajectories can intersect even if their points are far away from each other, when the objects move fast enough. The intersection is computed by the function POINTSINTERSECT(*trajectoryA, trajectoryB, pointA, pointB*). If the norm is greater than *maxDistance* and the neighboring lines do not intersect, the respective frame is removed from the result set. The second constraint is checked by the function FRAMEDEFINEDONTRAJECTORY(*trajectory, frameNumber*), which compares the frame-range of *trajectory* to the input *frameNumber*.

DISTANCE BETWEEN OBJECTS: SMALL:

Algorithm 3.2.7: CLOSETRAJECTORIES(Array trajectories, maxDistance)

```
local Set resultFrames ← GETFRAMESFROMRANGE(STOREDFRAMES(trajectories[0]))

for i ← 1 to SIZEOF(trajectories) − 1
  for each frame in resultFrames
    if FRAMEDEFINEDONTRAJECTORY(frame, trajectories[i])
      then
        for j ← 0 to i − 1
          local pointA ← POSITIONFORFRAME(frame, trajectories[i])
          local pointB ← POSITIONFORFRAME(frame, trajectories[j])
          local norm ← NORM(pointA.x-pointB.x, pointA.y-pointB.y)
          bool intersect ← POINTSINTERSECT(trajectories[i],trajectories[j],pointA,pointB)
          if not intersect and norm >maxDistance
            then DELETEFROMSET(frame)
      else DELETEFROMSET(frame)

return (setOfFrames)
```

Distance between Objects: Big

The pattern *Distance between Objects: Big* finds frames in which objects are far away from each other. This functionality is achieved by algorithm 3.2.8. The implementation is analog to that of algorithm 3.2.7. Here the variable *minDistance* describes the minimum distance of the objects.

Pattern-Implementation: Distance between Objects: Big

DISTANCE BETWEEN OBJECTS: BIG:

Algorithm 3.2.8: TRAJECTORIESFARAWAY(Array trajectories, minDistance)

```
local Set resultFrames ← GETFRAMESFROMRANGE(STOREDFRAMES(trajectories[0]))

for i ← 1 to SIZEOF(trajectories) − 1
  for each frame in resultFrames
    if FRAMEDEFINEDONTRAJECTORY(frame, trajectories[i])
      then
        for j ← 0 to i − 1
          local pointA ← POSITIONFORFRAME(frame, trajectories[i])
          local pointB ← POSITIONFORFRAME(frame, trajectories[j])
          local norm ← NORM(pointA.x-pointB.x, pointA.y-pointB.y)
          if norm <minDistance
            then DELETEFROMSET(frame)
      else DELETEFROMSET(frame)

return (setOfFrames)
```

Distance between Objects: Increases

The pattern *Distance between Objects: Increases* finds frames where two objects move away from each other. Its implementation consists of two main steps:

Pattern-Implementation: Distance between Objects: Increases

1. Compute intersecting range: First the range is computed where both trajectories T_A and T_B are defined. The remaining points of trajectory T_A are stored in the array $points_{T_A}$ and those of trajectory T_B are stored in $points_{T_B}$.

   Intersecting Range

2. Recursive Call: The recursive function

   Recursive Function

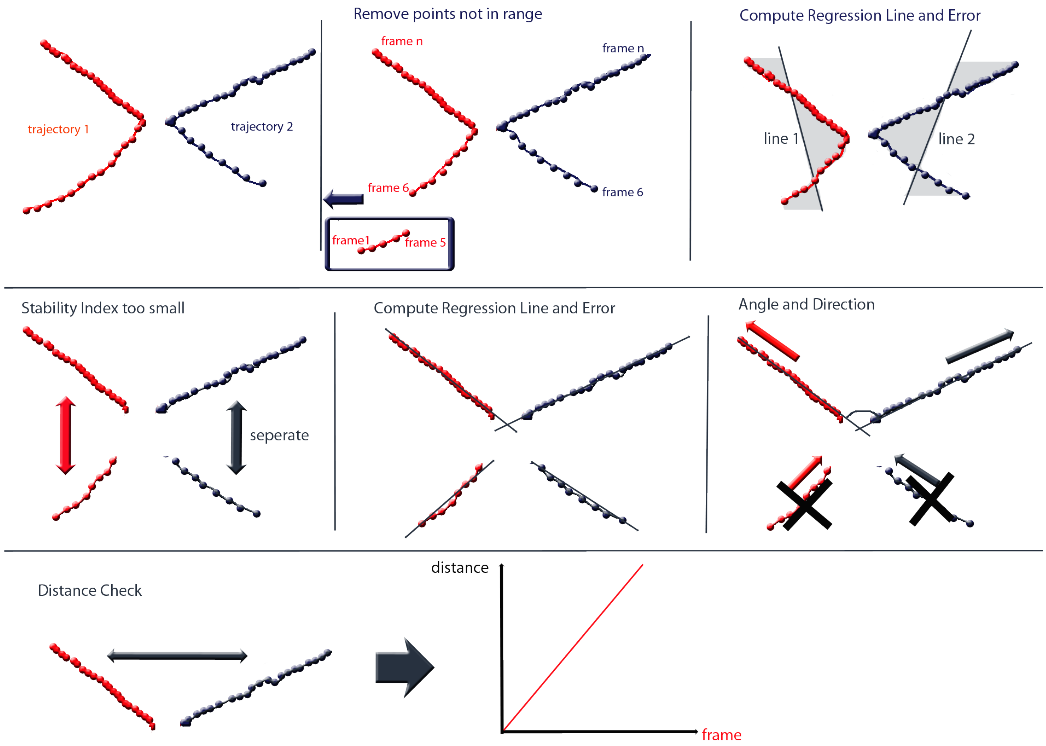

Figure 3.8: Shows steps performed by *Distance between Objects: Increases*-Pattern.

DISTANCEINCREASES($points_{T_A}$,$points_{T_B}$,*angle*,*precision*,*range*) is called. The *precision* value can be adjusted by the user. It defines the minimum number of frames into which the point arrays can be split to get higher accuracy. The *angle* defines the minimum angle by which the trajectories should deviate and *range* is the frame range on which both trajectories are defined. This function returns the frames where the two selected objects move away from each other.

Implementation of the Recursive Function

The recursive function described by algorithm 3.2.9 and visualized in figure 3.8 performs the following steps:

Linear Regression

1. Approximating Trajectories by Linear Regression: In a first step the regression lines of the points of trajectories T_A and T_B are computed (Cramer and Kamps [2008]). This step is performed by the function INITREGRESSIONLINEWITHPOINTS(). The linear re-

DISTANCE BETWEEN OBJECTS: INCREASES:

Algorithm 3.2.9: DISTANCEINCREASES(Array points$_{T_A}$,points$_{T_B}$,angle,precision,range)

```
local Set resultSet ← ∅
if SIZEOF(points_TA)<precision
  then return (resultSet)

local RegressionLine lineA ← INITREGRESSIONLINEWITHPOINTS(points_TA)
local RegressionLine lineB ← INITREGRESSIONLINEWITHPOINTS(points_TB)
local errorA ← STABILITYINDEX(lineA)
local errorB ← STABILITYINDEX(lineB)

if errorA <MAX_ERROR or errorB <MAX_ERROR
  then
  { local errorFrame
  { if errorA<errorB
  {   then errorFrame ← MEANVALUEFILTERING(points_TA)
  {   else errorFrame ← MEANVALUEFILTERING(points_TB)
  { local Range rangeI ← NEWRANGEFIRST(range, errorFrame)
  { local Range rangeII ← NEWRANGELAST(range, errorFrame)
  { local points_TA1 ← GETFIRSTHALFOFPOINTS(points_TA, errorFrame)
  { local points_TA2 ← GETLASTHALFOFPOINTS(points_TA, errorFrame)
  { local points_TB1 ← GETFIRSTHALFOFPOINTS(points_TB, errorFrame)
  { local points_TB2 ← GETLASTHALFOFPOINTS(points_TB, errorFrame)
  { resultSet ← DISTANCEINCREASES(points_TA1, points_TB1, angle, precision)
  { resultSet ← resultSet ∪ DISTANCEINCREASES(points_TA2, points_TB2, angle, precision)
  { return (resultSet)
  else
  { local realAngle ← GETANGLEFROMLINES(lineA, lineB)
  { local Array distanceHistrogram
  {
  { for i ← 0 to SIZEOF(points_TA)
  {   do
  {   { distanceHistogramm[i] ←
  {   {     NORM(points_TA[i].x-points_TB[i].x, points_TA[i].y-points_TB[i].y)
  {
  { local RegressionLine distance ←
  {     INITREGRESSIONLINEWITHPOINTS(distanceHistorgram)
  { if distance.slope >0 and realAngle≥ angle
  {   then resultSet ← resultSet ∪ GETFRAMESFROMRANGE(range)
return (resultSet)
```

gression is used to evaluate if the trajectories deviate by a certain angle.

Stability Index

2. Error Computation and Recursion: The stability index R is computed by the function STABILITYINDEX(). R is a value between zero and one, the higher this value is, the better the approximation.

$$R = 1 - \left(\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \right) \tag{3.5}$$

$$\hat{y}_i = \hat{a} + \hat{b} \cdot x_i \tag{3.6}$$

The y_i denote the y-values of the trajectory-points and n is the total number of points used for regression [Brete]. $\hat{a}$ and $\hat{b}$ constitute the regression-line model and $\bar{y}$ denotes the mean-value of the y-coordinates. According to Brete [2004] there is no defined standard maximal error value. Thus I tested the data for several error values on noisy and non-noisy video material. An error value that excepts both can not exceed 0.85. Should the resulting error values drop below this predefined threshold *MAX_ERROR* the point arrays are split, to achieve higher precision by performing a recursive call and by this approximating the trajectory by two regression lines. The point where the trajectories are split is computed by a mean value filtering of the point array with the smaller stability index. This process is visualized in figure 3.9. In each step of the filtering the point which has the smallest distance to its orthogonal projection on to the line connecting its two neighboring points is removed until only one interior trajectory point is left. This is the point where the trajectories are split. The function DISTANCEINCREASES() is then called on the defined subarrays, which are computed by the functions GETFIRSTHALFOFPOINTS(), GETLASTHALFOFPOINTS(), and NEWRANGE()-functions.

Mean Value Filtering

Recursive Call

Angle and Distance

3. Angle and Distance: If the error values of the linear regression are admissible the angle between the trajectories in movement direction is computed and tested against the input angle. Second a distance histogram of the points in the arrays points$_{T_A}$ and points$_{T_B}$ is created and a linear regression is performed on the distances. If the slope s of this regres-

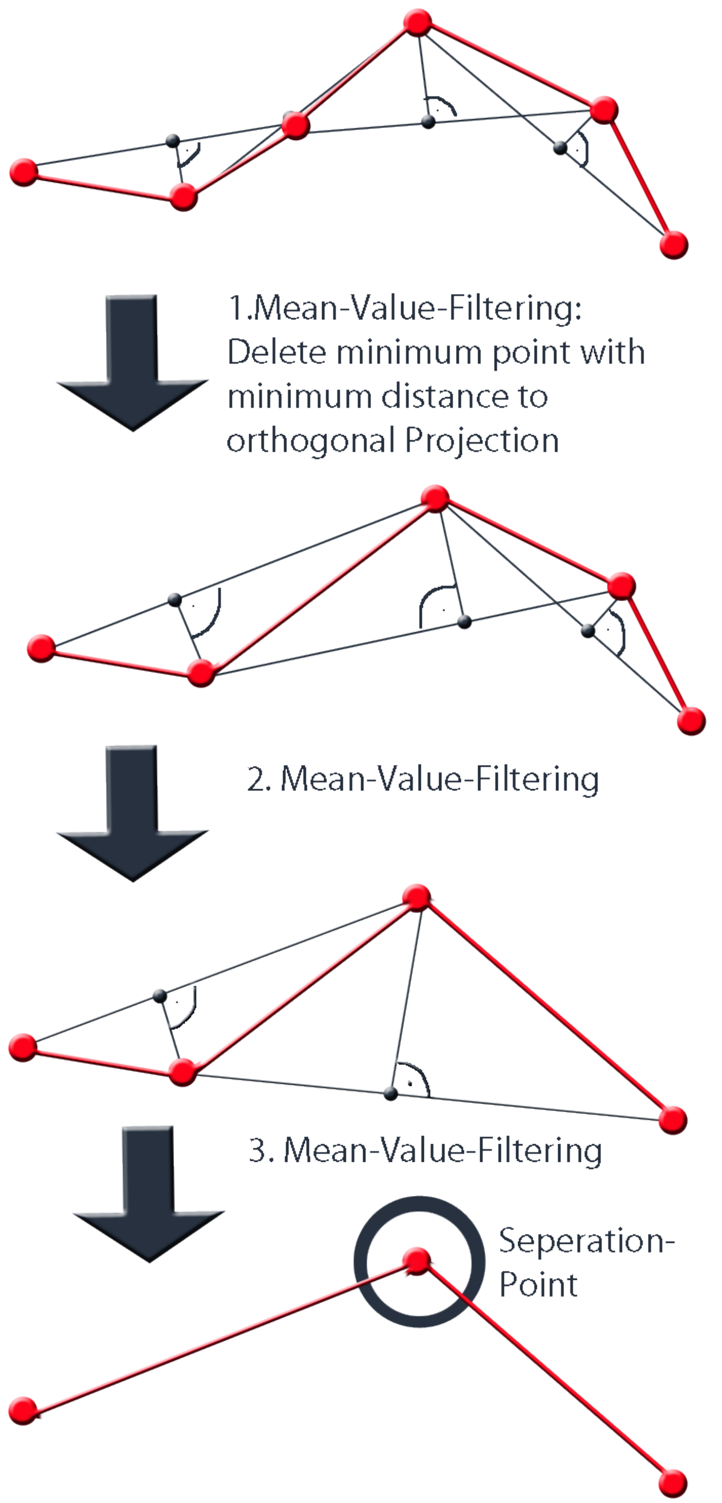

Figure 3.9: Steps performed by the function MEANVALUEFILTERING(*Array points*). To find the largest extent, step by step the point with the smallest distance to its orthogonal projection on the connecting line of the neighboring points is removed. In the last step, the interior point denotes the separation point.

sion line is $s \in \mathbb{R}^+$, the distance between the two trajectories increases with ascending frame numbers. If both tests are successful the frames in the range where both trajectories are defined can be added to the result.

Objects Meet

Pattern-Implementation: Objects Meet

With the pattern *Objects Meet* the user is able to find frames where all objects are within intersection distance. The intersection distance is defined by 5% of the movie size width, which shows satisfying results. To compute an exact intersection tolerance one would need to compute the average object size in a video. Algorithm 3.2.10 returns the frames where all trajectories in the input array intersect. The user has the possibility to select a dependance value. If she chooses T_1 algo-

OBJECTS MEET:

Algorithm 3.2.10: TRAJECTORIESINTRESECT(Array trajectories, dependance)

```
if dependance == T1
  then
  {return (CLOSETRAJECTORIES(trajectories,movieSize.width*INTERSECTION_RATIO))
  else
  local Set arraysOfFrames ← ∅
  local Array frames ← INITWITHCAPACITY(SIZEOF(trajectories))
  arraysOfFrames ←
       GETFRAMESOFINTERSECTINGPOINTS(frames, trajectories, -1, 0)
  local Set resultSet ← ∅
  for each array in arraysOfFrames
    do
    for each set in array
      do
      {resultSet ← resultSet ∪ set

  return (resultSet)
```

rithm 3.2.10 returns all frames, where the trajectories intersect in the same frame. In this case the function CLOSETRAJECTORIES(), which was described above, is

OBJECTS MEET:

Algorithm 3.2.11: GETFRAMESOFINTERSECTINGPOINTS(Array frames,t,tNr,pointNr)

```
local Set resultSet ← ∅
if not tNr == −1
  then
   if tNr == SIZEOF(t)
     then return (SETWITHOBJECT(frames))
   local currentFrame ← pNr + STOREDFRAMES(t[tNr]).location
   for i ← 0 to t − 1
     do
      local Point point ← POSITIONFORFRAME(t[i], frames[i])
      local Point thisPoint ← POSITIONFORFRAME(t[tNr], currentFrame)
      bool intersect ← POINTSINTERSECT(t[i], t[tNr], point, thisPoint)if not intersect
        then return (∅)
      frames[i] ← currentFrame

tNr++
if tNr<SIZEOF(t)
  then
   for i ← 0 to STOREDFRAMES(t[tNr]).length
     do
      resultFrames ←
           resultFrames ∪ GETFRAMESOFINTERSECTINGPOINTS(frames, t, tNr, i)
  else resultFrames ←
        resultFrames ∪ GETFRAMESOFINTERSECTINGPOINTS(frames, t, tNr, 0)
return (resultSet)
```

called for the intersection distance. When she picks T_2 algorithm 3.2.11 is called. This algorithm searches for all trajectory intersections independent of the frame number. In the following I will describe the steps of the function GETFRAMESOFINTERSECTINGPOINTS(*frames*, *t*, *tNr*, *pointNr*). This algorithm functions in a back-tracking-like manner. The first call gets as input an empty array *frames*, that contains as many entries as the array *t*, which holds the defined trajectories. So each entry in *frames* is reserved for a frame number of its corresponding trajectory in *t*. At each call the function checks if the point of trajectory *t[tNr]* in frame *pointNr*+STOREDFRAMES(*t[tNr]*).*location* is in intersection distance to all other points which have already been checked against their preceding trajectories. If the point at *currentFrame* is close to all the points

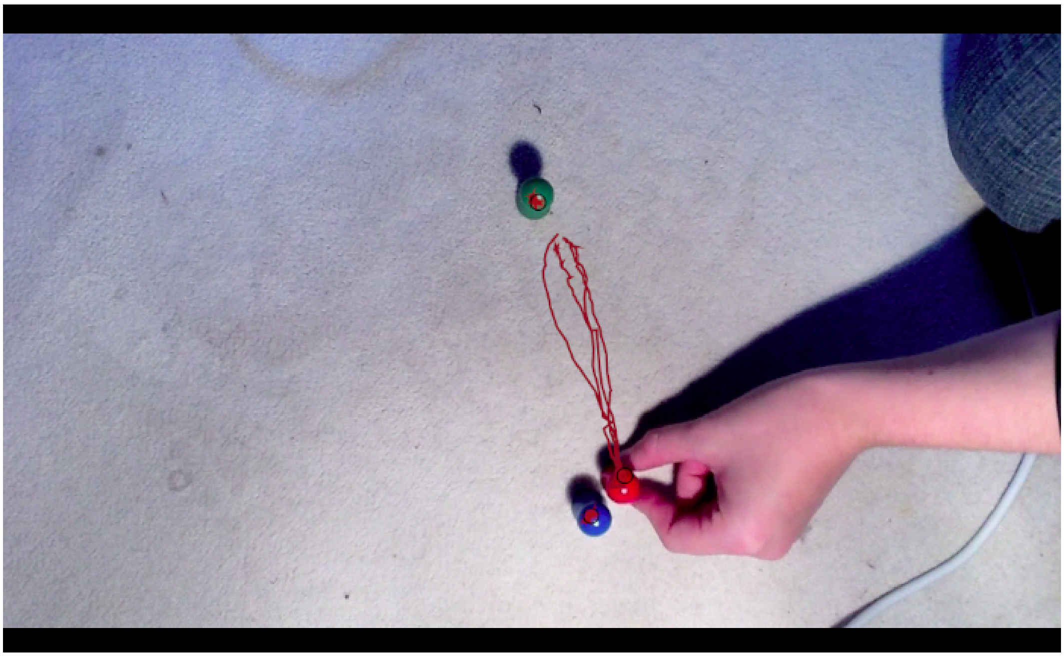

Figure 3.10: Trajectories of pawns. The red pawn moves between the green pawn and the blue one. This is regonized by the pattern *Object meets several other objects*, when the red pawn is selected as special object.

corresponding to the frames in the array *frames*, which are defined up to index *tNr-1* the new frame is inserted into *frames[tNr]*. Should this not be the case $\emptyset$ is returned, and this branch is abolished. To find all intersections GETFRAMESOFINTERSECTINGPOINTS() is called on every frame of every trajectory. But since branches are discarded early the number of function calls is reduced strongly. The filled array *frames* is added to the result set if for each trajectory t_x a point was found that is close to all other locations of the trajectories t_i at the respective frame *frame*[t_i].

Object meets several other Objects

Pattern Implementation: Object meets several other Objects

The pattern *Object meets several other Objects* finds frames, where the trajectory of a special predefined object intersects the trajectories of all other selected objects. An example is depicted in figure 3.10. Algorithm 3.2.12 describes this functionality. The input *index* is an integer value, which denotes the index of the selected object. To find the respective frames algorithm 3.2.12 calls the function CLOSETRAJECTORIES() on each couple of trajectories *(trajectories[index], trajectories[i])*, with $i \in \{0, \ldots, \text{SIZEOF}(\textit{trajectories})\} \backslash \{\textit{index}\}$.

OBJECT MEETS SEVERAL OTHER OBJECTS:

Algorithm 3.2.12: TRAJECTORYINTRESECTSTRAJECTORIES(Array trajectories, index)

```
local Set resultSet ← ∅
for i ← 0 to SIZEOF(trajectories)
  do
    if not i == index
      then
        local Array inputArray ← ARRAYWITHOBJECTS(trajecories[index], trajectories[i])
        resultSet ← resultSet ∪
            CLOSETRAJECTORIES(inputArray, movieSize.width*INTERSECTION_RATIO)
return (resultSet)
```

Distance between two Objects increases after a close Motion

Pattern Implementation: Distance between two Objects increases after a close Motion

The pattern *Distance between two Objects increases after a close Motion* identifies scenarios like the lost-luggage event. An example is depicted in figure 3.11. At first the two objects with trajectories t_A and t_B are within distance *dist* and at a certain frame they deviate from each other. This functionality is described by algorithm 3.2.13. The user has several options. She is able to adjust precision values, as in the function DISTANCEINCREASES(), she can define which percentage of frames shall be close, before objects deviate with the value *lenClose*, and she can choose if the left object needs to be still after it was abandoned with the parameter *isStill*. Additionally the stability index for the function call DISTANCEINCREASES() can be adjusted in this pattern, since a strict definition of the line-precision could refuse correct results. In a first step the algorithm extracts all frames where the objects deviate

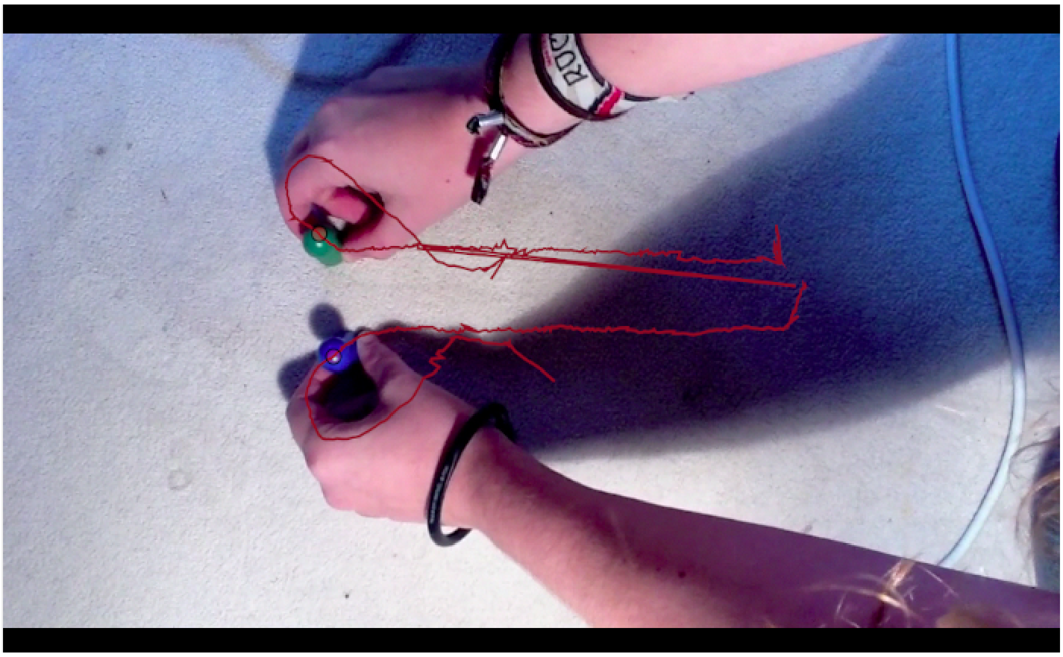

Figure 3.11: Object trajectories of two pawns where the distance between the two objects increases after they moved together.

from each other and are within distance *dist*. The function GETNUMBEROFFRAMESCLOSEBEFOREFRAME() calculates the number of frames in which the input trajectories t_A and t_B are close before frame *frame*. After checking if the objects where close before they increased distances, the optional check for still objects is done. The function STILLAFTERFRAME() checks the variance of the trajectory points, after the deviation.

Objects are visible in the same Frame

Pattern Implementation: Objects are visible in the same Frame

The pattern *Objects are visible in the same Frame* enables the user to find all frames, in which all selected objects are visible. These frames are returned by algorithm 3.2.14. by intersecting all frame sets of the single trajectories.

DISTANCE BETWEEN TWO OBJECTS INCREASES AFTER A CLOSE MOTION:

Algorithm 3.2.13: DEVIATEAFTERCLOSEMOTION(t_A, t_B, isStill,precision,dist,lenClose)

local Set resultSet ← $\emptyset$
local Range range ← INTERSECTRANGE(t_A, t_B)
local Array points$_{T_A}$ ← POINTSINRANGE(t_A, range)
local Array points$_{T_B}$ ← POINTSINRANGE(t_B, range)
local Array input ← ARRAYWITHOBJECTS(t_A, t_B)

resultSet ← DISTANCEINCREASES(points$_{T_A}$, points$_{T_B}$, angle, precision, range)
resultSet ← resultSet ∩ CLOSETRAJECTORIES(input, dist)
for each frame **in** resultSet
  do
    local int count ← GETNUMBEROFFRAMESCLOSEBEFOREFRAME(frame, t_A, t_B)
    if count<lenClose
      then REMOVEFROMSET(resultSet, frame)
      else if isStill
      then
        if not ISSTILLAFTERFRAME(t_A, frame) **and not** ISSTILLAFTERFRAME(t_B, frame)
          then REMOVEFROMSET(resultSet, frame)

return (resultSet)

OBJECTS ARE VISIBLE IN THE SAME FRAME:

Algorithm 3.2.14: VISIBLEINSAMEFRAME(Array trajectories)

local Set resultSet ← FRAMESFROMRANGE(STOREDFRAMES(trajectories[0]))
for $i \leftarrow 1$ **to** SIZEOF(trajectories) $- 1$
  do resultSet ← resultSet ∩ FRAMESFROMRANGE(STOREDFRAMES(trajectories[i]))
return (resultSet)

Objects have Parallel Trajectories

Pattern Implementation: Objects have Parallel Trajectories

The pattern *Objects have Parallel Trajectories* searches for frames in which the selected trajectories have similar characteristics (see figure 3.12). To evaluate the resemblance of each two trajectories, the system performs a scale invariant mesh registration on the object-trajectories. The algorithm 3.2.15 functions analog to algorithm 3.2.9 with the call DISTANCEINCREASES(), which has been explained in

Figure 3.12: To people moving parallel to each other. Most frames are recognized by the pattern-recognizer *Objects have parallel trajectories.*(See chapter 4 for more detail.)

Mesh registration via SICP

detail in one of the previous sections. The mesh registration is computed via the scale invariant closest point algorithm (SICP) (Shaoyi Du and You [ICIP 2007]). Main SICP operations are depicted in figure 3.14. As input the SICP-algorithm gets two point sets $M = \{m_i\}_{i=1}^N$ and $P = \{p_i\}_{i=1}^N$, where the points m_i and p_i are two dimensional points in the (x,y)-space and N is the number of points in each set. Since I only regard points where both trajectories are defined, both sets have the same size. The goal of the SICP algorithm is to find the minimum of the following least squares problem:

$$\text{sicpError} = \min_{s,R,t} \sum_{i=1}^{N} \|(RSp_i + t) - m_i\|_2^2 \tag{3.7}$$

S denotes the scaling matrix, where the scaling factor shall not be greater than 3. R represents the rotation matrix and t describes the translation vector. The mesh registration is performed by iteratively converging against the solution which minimizes equation (3.7). All iterative steps are described in detail in [Du:2007]. After the SIPC computation is performed, the SICP properties, like rotation and scaling matrix and the SICP error (3.7) are checked. If these values are below the predefined thresholds, the corresponding frames can be added to the result. Otherwise the arrays are split at the point which maximizes the SICP-error. This point is computed by the function GETINDEXOFINTERIORMAXERRORCONTRIBUTION(). The

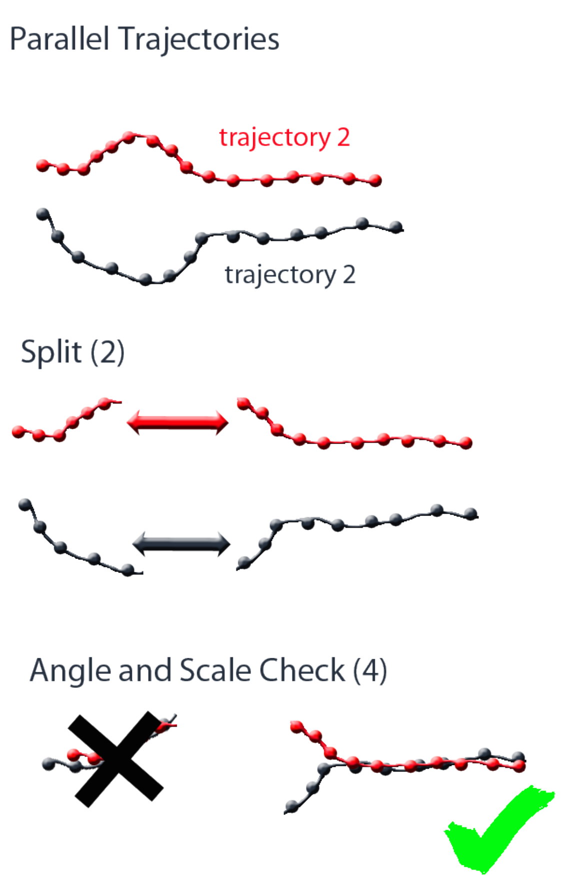

Figure 3.13: Parallel Trajectories Algorithm. Shows two trajectories which are checked to be parallel. After sicp a split is performed because the error is too large. The second sicp discards the left part of the trajectory because of the rotation by 45 degree.

course of the recognition algorithm is visualized in figure 3.13.

An Object Moves from one Object to an Other

Pattern-Implementation: An Object Moves from one Object to an Other

The pattern *An Object Moves from one Object to an Other* identifies events like passes, where a defined object is passed between the other objects. Algorithm 3.2.16 searches for frames, where a predefined object arrives at or leaves one of the other objects in the array *trajectories*. The integer value *index* denotes the index of the passed object

Objects have Parallel Trajectories:

Algorithm 3.2.15: FRAMESFROMSICP(Array points$_{T_A}$,points$_{T_B}$,angle,precision,range)

local Set resultSet ← ∅
if SIZEOF(points$_{T_A}$)<precision
then return (resultSet)

local **SICP** sicp ← INITSICPWITHPOINTS(points$_{T_A}$, points$_{T_B}$)

if sicp.scaleCoefficient >3 **or** sicp.scaleCoefficient <0.3
or sicp.error >SICP_ERROR*movieSize.width
or sicp.rotationAngle >angle
then
local errorFrame ← sicp.GETINDEXOFINTERIORMAXERRORCONTRIBUTION()
local **Range** rangeI ← NEWRANGEFIRST(range, errorFrame)
local **Range** rangeII ← NEWRANGELAST(range, errorFrame)
local points$_{T_{A1}}$ ← GETFIRSTHALFOFPOINTS(points$_{T_A}$, errorFrame)
local points$_{T_{A2}}$ ← GETLASTHALFOFPOINTS(points$_{T_A}$, errorFrame)
local points$_{T_{B1}}$ ← GETFIRSTHALFOFPOINTS(points$_{T_B}$, errorFrame)
local points$_{T_{B2}}$ ← GETLASTHALFOFPOINTS(points$_{T_B}$, errorFrame)
resultSet ← FRAMESFROMSICP(points$_{T_{A1}}$, points$_{T_{B1}}$, angle, precision)
resultSet ← resultSet ∪ FRAMESFROMSICP(points$_{T_{A2}}$, points$_{T_{B2}}$, angle, precision)
return (resultSet)
else
then resultSet ← resultSet ∪ GETFRAMESFROMRANGE(range)**return** (resultSet)

and the variable *mLength* describes the length of the input video in frames. As in DEVIATEAFTERCLOSEMOTION() the stability index is adjustable by the user to avoid too strong filtering. First the algorithm computes all frames where one of the trajectories is close to the trajectory *trajectories[index]* and where the distance increases after this frame. Since I want to filter out the first and the last frame of each trajectory intersection the frames are stored in the array *frames* in ascending order, by writing them into a data-structure *trajectoryFrames=(frameNumber, trajectoryNumber)*. The superfluous frames are deleted by the function DELETEIFPREVIOUSANDCONSECUTIVEFROMSAMETRAJECTORY(). GETSETOFFRAMESFROMTRAJECTORYFRAMEARRAY() invokes that the set of resulting frames is computed from the array containing the *trajectoryFrames*.

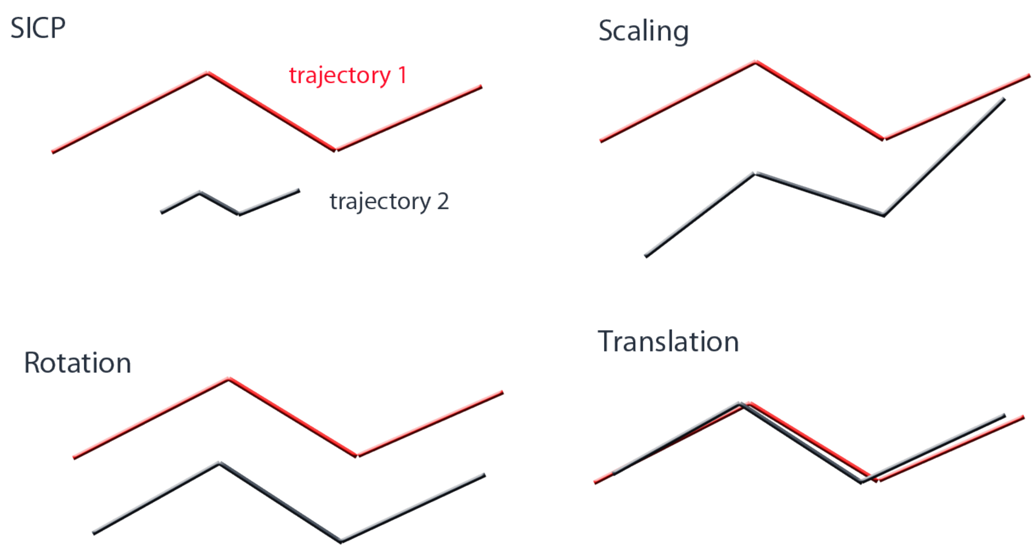

Figure 3.14: Shows the three operations performed by the sicp-algorithm: scaling, rotation, and translation.

3.2.5 Direction and Velocity

Cluster: Direction and Velocity

In the areas behavioral research, visual surveillance, and forensic analysis a forth cluster, *Direction and Velocity*, is of interest. This cluster includes the patterns *Objects move in an opposing Direction compared to the average Object Motion*, *Velocity Differs from Own Average Velocity*, and *Velocity Differs from Average Velocity of all Trajectories*. An example is the *ADVISOR* (Siebel and Maybank [2004]), an automated visual surveillance system for metro stations, which automatically detects dangerous situations. One task of the *ADVISOR* is to perform crowd analysis, by detecting counter-flow (Siebel and Maybank [2004]), where people are moving against the main flow or in opposite directions in one-way paths. Also, Shobhit Saxena and Ma [2008] state that people moving in counter direction is an important pattern in crowd analysis. Furthermore, surveillance cameras exist that support counterflow detection like the CCTV Equipments from Karthik Energy Technologies (Kar). According to JBN [June 24, 2010] in traffic surveillance, motorists represent a threat, which should be recognized fast. These examples can be found by the cluster

Counter Flow

An Object Moves from one Object to an Other:

Algorithm 3.2.16: OBJECTPASSEDBETWEENOBJECTS(Array trajectories, index, mLength)

```
local Array intersectionArray ← INITWITHCAPACITY(SIZEOF(trajectories))
for each t in trajectories
  do
  { if t==trajectories[index]
      then continue

    local Range range ← INTERSECTRANGE(t, trajectories[index])
    local Array points_TA ← POINTSINRANGE(t, range)
    local Array points_TB ← POINTSINRANGE(trajectories[index], range)
    local Array input ← ARRAYWITHOBJECTS(t, trajectories[index])

    intersectionArray[i] ←
        DISTANCEINCREASES(points_TA, points_TB, angle, precision, range)
    intersectionArray[i] ← intersectionArray[i] ∩
        CLOSETRAJECTORIES(input, movieSize.width*INTERSECTION RATIO)
local Array frames
for f ← 0 to mLength
  do
  { for t ← 0 to SIZEOF(trajectories)
      do
      { if f ∈ intersectionArray[t]
          then
          { local trajectoryFrame tF ← INITWITHFRAMEANDTRAJECTORY(f, t)
            frames.ADDOBJECT(tF)

for i ← 0 to SIZEOF(frames)
  do
  {DELETEIFPREVIOUSANDCONSECUTIVEFROMSAMETRAJECTORY(i, frames)
local Set resultSet ← GETSETOFFRAMESFROMTRAJECTORYFRAMEARRAY(frames)
return (resultSet)
```

Velocity

Objects move in an opposing Direction compared to the average Object Motion. Robert Bodor and Papanikolopoulos [2003] developed methods to detect situations where people are in need. They define running or moving erratically and loitering as suspicious behavior. According to Saunier and Sayed [2007], speed detection in traffic surveillance is an important task. Also, an animal being chased by an other animals or when animals move faster or slower than usual are interesting events. These situations can be found with the two patterns *Velocity Differs from Own Average Velocity*,

Figure 3.15: Video from the PETS-workshop 2009. Object trajectories of 3 people. One is moving into an opposing direction.

and *Velocity Differs from Average Velocity of all Trajectories*.

Objects move in an opposing Direction compared to the average Object Motion

Pattern-Implementation: Objects move in an opposing Direction compared to the average Object Motion

The implementation of the pattern *Objects move in an opposing Direction compared to the average Object Motion* finds frames, in which a certain percentage of trajectories deviates by at least ninety degrees from the average course. An example is depicted in figure 3.15. Algorithm 3.2.17 identifies this trajectory-behavior. The user has the ability to adjust the minimum percentage value. First the algorithm calculates the trajectory angles for each frame and stores them in the two dimensional array *angleArrays[trajectory][frame]*. Afterwards *angleArrays* contains the angles of each trajectory at every frame. This step is performed by the function GETARRAYSOFANGLESFROMTRAJECTORIES(Array *trajectories*). Second the average direction and the average filtered direction of the trajectories are computed for each frame. Resulting values for all frames are stored in the array *averageDirection[frameNr]* by calling the function GETAVERAGEDIRECTIONINROW(*angleArray, frameNr*). Algorithm 3.2.17 mere shows the computation of the mean direction, since the filter-checks occur in the same manner. The filtering takes trajectory-angles of its two neighboring frames and computes the mean direction of the trajectory in these three frames. Applying filters reduces the presumption of false positives due to noisy data or when objects move very slow. Next the percentage of trajectories, which's directional aberration is greater than ninety degrees from the mean direction, is calculated. If this value exceeds the user-defined percentage the respective frame is added to the results. When a trajectory is identified, having differing directions in frame i, the mean-direction is updated by calling UPDATEAVERAGEDIRECTION(*averageDirection[i], trajectoryNr*), which recomputes the average direction by not considering the direction at index *trajectoryNr*.

Velocity Differs from Own Average Velocity

Velocity Differs from Own Average Velocity

The pattern *Velocity Differs from Own Average Velocity* searches frames, where an object's pace differs from its own average speed. Figure 3.16 shows a car that gets slower when turning left. This is a situation that is recognized by the algorithm. For each trajectory algorithm 3.2.18 first computes the mean value in $\frac{\text{distance}}{\text{frame}}$ of all trajectory-lines l = (POSITIONFORFRAME(*trajectories[i], j*), POSITIONFORFRAME(*trajectories[i], j+1*)), which is achieved by calling GETAVERAGEVELOCITYFROMTRAJECTORY(*trajectory*). By adjusting the percent value the user defines the tolerance by which the pace per frame may differ from the mean velocity. Each trajectory section l is tested against the tolerance values *deviationUp* and *deviationDown* and added to the result if the respective velocity exceeds these boundaries.

Opposing Direction:

Algorithm 3.2.17: OPPOSINGDIRECTION(Array trajectories, percent)

```
local Set resultSet ← ∅
local Range range ← UNIONRANGE(trajectories)
local Array angleArrays ← GETARRAYSOFANGLESFROMTRAJECTORIES(trajectories)
local Array averageDirection ← INITWITHCAPACITY(range.length)
for i ← 0 to range.length
  do averageDirection[i] ← GETAVERAGEDIRECTIONINROW(angleArrays, i)
for i ← 0 to range.length
  do
  { local int count ← NUMBEROFDEFINEDOBJECTSAT(i)
  { local int countDeviating ← 0
  { for t ← 0 to SIZEOF(angleArrays)
  {   do
  {   { if not TRAJECTORYDEFINEDINFRAME(t, i)
  {   {   then continue
  {   { if averageDirection[i]-angleArrays[t][i] >90
  {   {   then
  {   {   { countDeviate ++
  {   {   { UPDATEAVERAGEDIRECTION(averageDirection[i], t)
  { if countDeviate ≤ count*percent*0.01
  {   then resultSet. < ADDOBJECT(i)

return (resultSet)
```

Velocity Differs from Average Velocity of all Trajectories

The pattern *Velocity Differs from Average Velocity of all Trajectories* selects frames, where an object's pace differs from the average velocity of the other trajectories. Algorithm 3.2.19 operates similar to algorithm 3.2.18. The additional input variable *frames* is an integer value that denotes the number of frames held by the input video. For each frame the average pace in $\frac{\text{distance}}{\text{frame}}$ is computed calling AVERAGEVELOCITYINFRAME(Array *trajectories*, int *frameNr*). Finding a trajectory, which exceeds or goes below the tolerance value specified in percentage by the user at a certain frame, adds this respective frame to the result.

Velocity Differs from Own Average Velocity

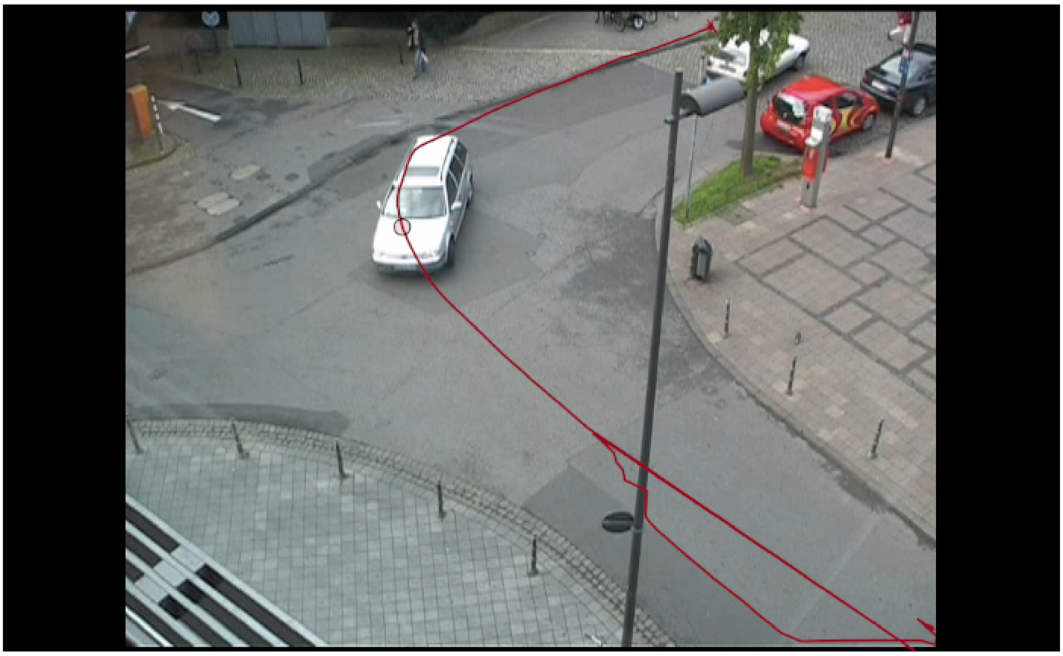

Figure 3.16: Object trajectories of a car, which is slower when turning left. (Video by Karrer et al.)

3.3 User Interface

User Interface

A User Interface for event detection software which benefits video analysts needs to provide easy access, overview of the discovered events, and a comprehensible depiction of the search criteria. Before presenting the user interface, designed for event detection, I will adumbrate the state of the art video browsing in behavioral research and video ethnography and describe a new video annotation software which addresses this target group.

3.3.1 Video Browsing: State of the Art

Video Browsing and Analysis

At the moment, video browsing is accomplished on different levels of technical support. Benjamin Dennig, research manager at the GIM states that the GIM uses nearly no technical aid for video browsing. Fouse et al. recognized this gap (Adam S. Fouse and Hollan [2011a], Adam S. Fouse

VELOCITY DIFFERS FROM OWN AVERAGE VELOCITY:

Algorithm 3.2.18: VELOCITYDIFFERSFROMOWNVELOCITY(Array trajectories, percent)

```
local Set resultSet ← ∅
for i ← SIZEOF(trajectories)
  do
  { local float averageVelocity ←
        GETAVERAGEVELOCITYFROMTRAJECTORY(trajectories[i])
    local float deviationUp ← averageVelocity+averageVelocity*percent*0,01
    local float deviationDown ← averageVelocity-averageVelocity*percent*0,01
    local Range range ← STOREDFRAMES(trajectories[i])
    for j ← range.location to range.length-2
      do
      { if not TRAJECTORYDEFINEDINFRAME(trajectories[i], j)
          then continue
        local Point pointA ← POSITIONFORFRAME(trajectories[i], j)
        local Point pointB ← POSITIONFORFRAME(trajectories[i], j+1)
        local float norm ← NORM(pointA.x-pointB.x, pointA.y-pointB.y)
        if norm<deviationDown or norm>deviationUp
          then resultSet.ADDOBJECTS(j, j + 1)

return (resultSet)
```

VELOCITY DIFFERS FROM OWN AVERAGE VELOCITY:

Algorithm 3.2.19: VELOCITYDIFFFROMAVERAGEVELOCITY(trajectories, percent, frames)

```
local Set resultSet ← ∅
for i ← 0 to frames-1
  do
  { local float averageVelocity ← AVERAGEVELOCITYINFRAME(trajectories, i)
    local float deviationUp ← averageVelocity+averageVelocity*percent*0,01
    local float deviationDown ← averageVelocity-averageVelocity*percent*0,01
    for t ← 0 to SIZEOF(trajectories − 1)
      do
      { if not TRAJECTORYDEFINEDINFRAME(trajectories[t], i)
          then continue
        local Point pointA ← POSITIONFORFRAME(trajectories[i], j)
        local Point pointB ← POSITIONFORFRAME(trajectories[i], j+1)
        local float norm ← NORM(pointA.x-pointB.x, pointA.y-pointB.y)
        if norm<deviationDown or norm>deviationUp
          then resultSet.ADDOBJECTS(j, j + 1)
```

Figure 3.17: FinalCutPro User Interface. Fin [b]

and Hollan [2011b]). They indicate, that researchers cannot afford the time to see all the recorded video material. Therefore they propose *ChonoViz*, a video annotation software, which I will describe below.

Compacting Videos for Analysis

The analysis process at the GIM is build on three steps. In a first step videos are assorted and thematically ordered. For the resulting data video time codes are created, that contain the important scenes, i.e. which part of the video is cut out. Finally the video material is cut with software like *Final Cut* (Fin [a]) or *Adobe Premiere* (Pre), an example of the Final Cut Pro user interface is shown in figure 3.17.

Video Annotation

In behavioral research the benefits of tracking software and video annotation are exploited. Behavioral researcher Benjamin Zipser reports that their institute uses two kinds of software. For journalizing, interpretation and analysis of videos they use video annotation software like the *Observer* from Noldus (Obs [a]) or *Interact* from Mangold (Int). Userinterface examples are depicted in figure 3.18 and 3.19. Noticeable are the extensive multiple timeline visualizations, where important scenes are marked.

Object Tracking

The second group of software, which the Department of Behavioral Biology from the University of Münster applies, are tracking systems to automate test-

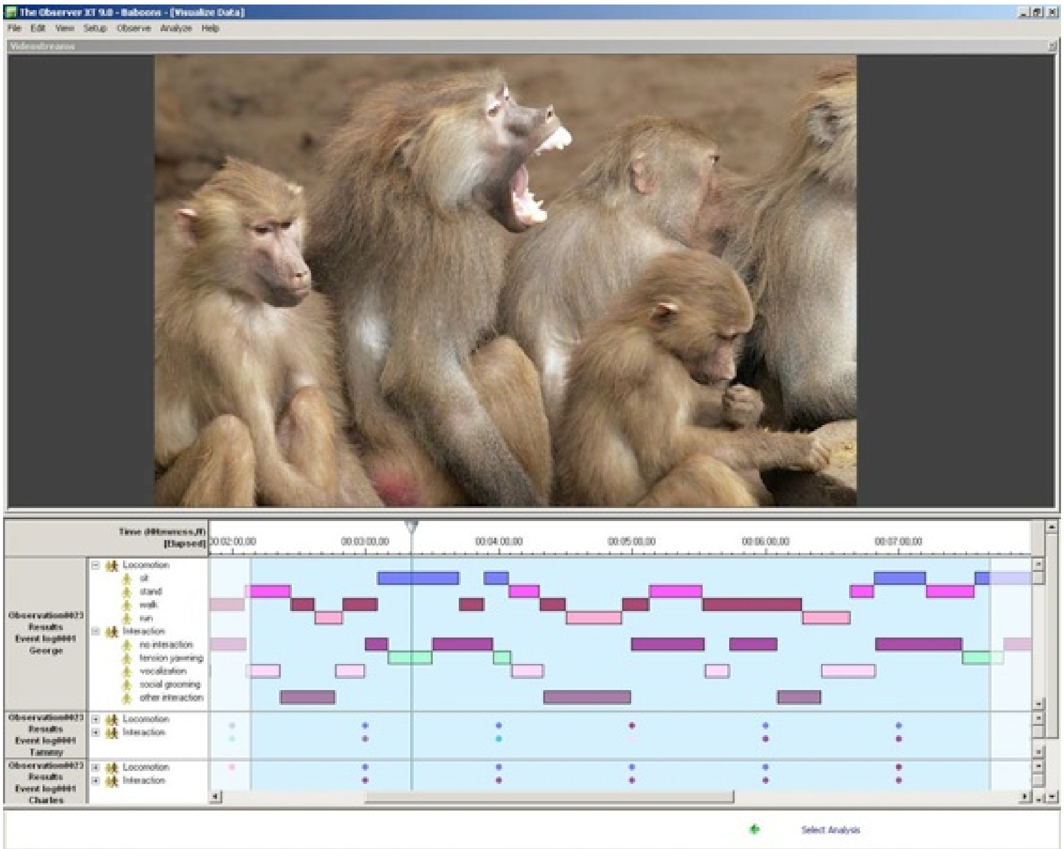

Figure 3.18: *Observer* from Noldus. Obs [b]

ing alike *AnyMaze* from Stoeling (Any). According to Benjamin Zipser this software enables them to find any test situation, where only one animal is involved. The user interface of *AnyMaze* is depicted in figure 3.20. Fouse et al. developed the software *ChronoViz*, a system to support visualization and analysis of time coded data (Adam S. Fouse and Hollan [2011a], Adam S. Fouse and Hollan [2011b]). Supporting observational research, their main target group is composed of researchers, behavioral scientists and ethnographers. Users are able to store various kinds of information in the video data. After data collection the time coded information is visualized in multiple timelines, as seen in figure 3.21. This enables the users to depict information in different categories. A quick access of the data snippets is achieved by showing popovers in the timeline when clicking on a 'time event' in the multiple timelines. *ChronoViz* was successfully tested in various domains (Adam S. Fouse

ChronoViz

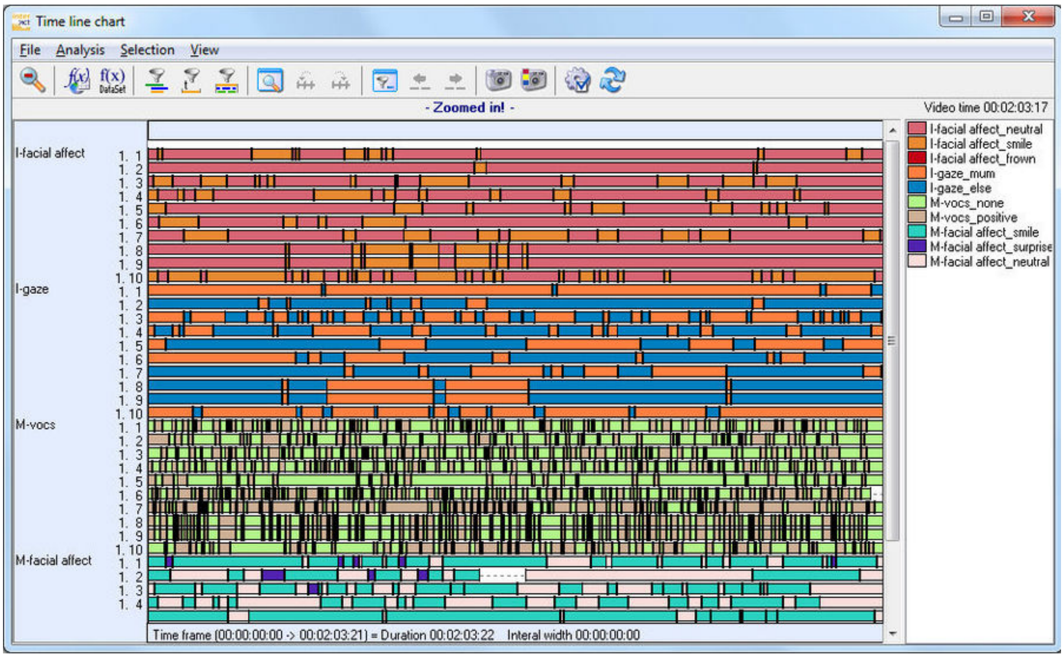

Figure 3.19: *Interact* from Mangold. Int

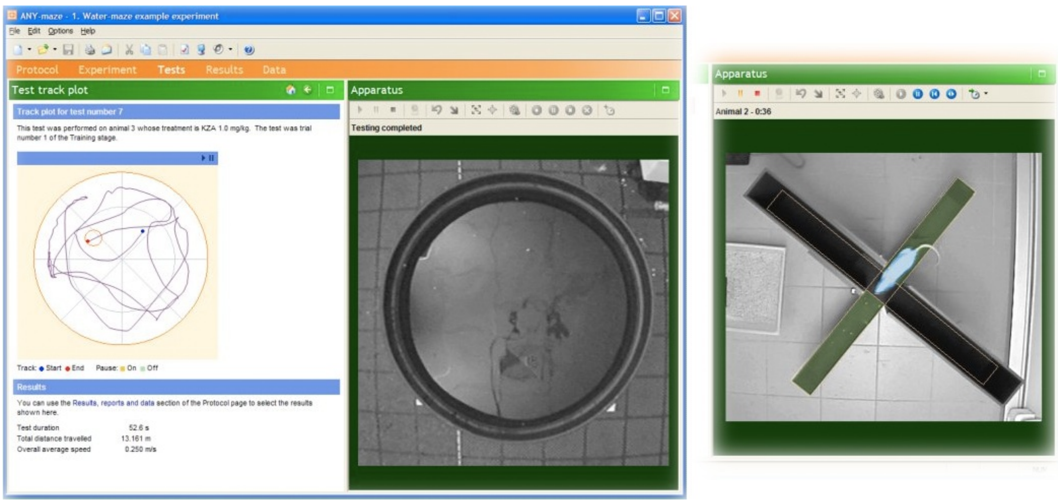

Figure 3.20: User-interface of AnyMaze from Stoeling. Any

and Hollan [2011b]). It shows that it allows fast and easy navigation and accelerates data collection and analysis.

Figure 3.21: Video annotation in ChronoViz. [Adam S. Fouse and Hollan [2011a]]

3.3.2 Visualizing and Navigating through Results

User Interface: Navigation and Results

Each named software type contains a standard video-player. Annotation software additionally offers options for multiple timelines, that contain information on the video frames, and tracking software provides features for automated testing, e.g. like viewing the object trajectories (see figure 3.20). Considering the described software types shows that the event detection extension for *DRAGON* should fulfill certain constraints for easy navigation through the results. As in each described system, the event detection extension will hold a standard video player. A predominant number of the named software types exploits multiple timelines for visualizing important data and helps the user structuring time coded data into

Timeline-Visualization

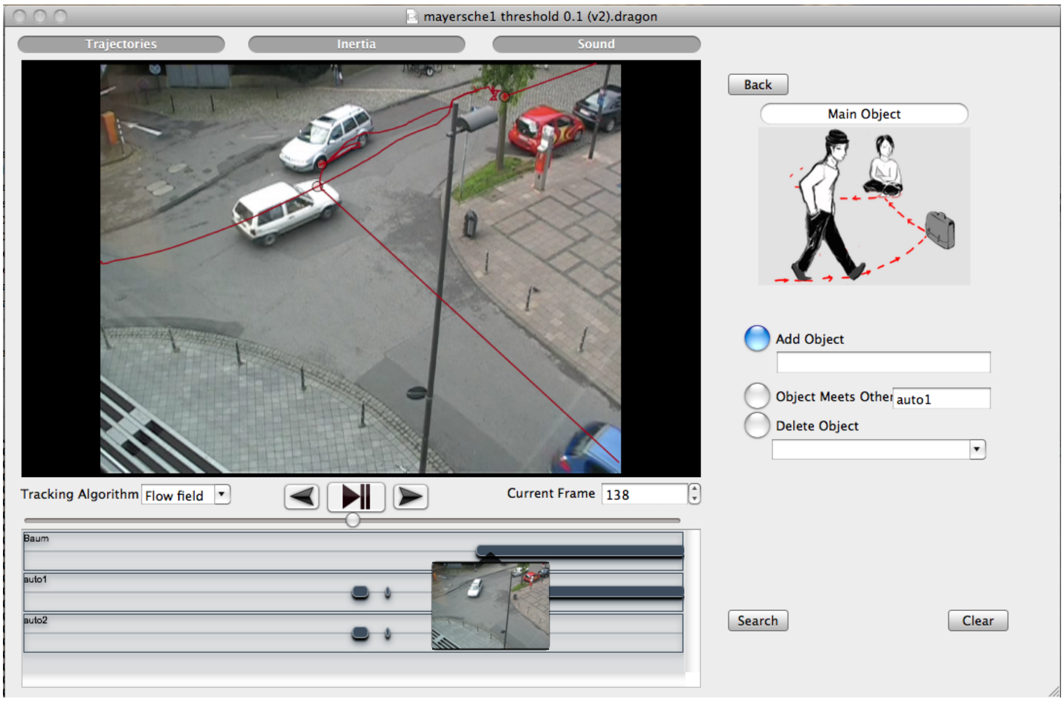

Figure 3.22: User Interface of event detection software where results are calculated.

Navigation

categories. Thus I will provide multiple timelines for all patterns where objects are analyzed independently, as seen in figure 3.22. Each object can be labeled and is assigned to its respective timeline. Patterns where the recognition is intertwined among the trajectories are depicted in a single timeline. These timelines contain colored ranges, which indicate the relevant frames. As in *ChronoViz*, the event detection system facilitates the user to easily access the content of the identified events by clicking on them and by this opening a popover showing a miniature of the respective frame. Navigation can be performed in three different accuracies. First, the standard video player offers common timeline navigation. Second, the in scene navigation provided by *DRAGON* allows the user to wind on a very fine grained level. Last, the user can step through the detected events by clicking on the left and right arrows located next to the play/pause button depicted in figure 3.22.

3.3.3 Pattern Selection Menu and Visualization

Pattern Visualization

Before the user can search for events she needs to select a pattern from seventeen choices. Since the menu mere contains keywords to describe the patterns, a simple textual description is not sufficient to enable a fast pattern selection. Thus each pattern is underlined with a comic-like image, describing a scenery of this pattern. The user can perceive the object constellation at a glance and decide if this pattern is suited for the event she seeks. According to [Cloud], a book on comic design by Scott Mc Cloud, even rough sketches should enable the user to understand the action clearly. They state that for simple actions only one image is required for the reader to understand the action. Furthermore the image-content is supported by motion-arrows which indicate the trajectory course. Dan B. Goldman and Seitz [2006]] state that motion arrows are often used by story board artists to describe and clarify motion paths. From the main menu the user can enter one of the four cardinal clusters *Areas*, *Objects Act*, *Objects Interact*, and *Direction and Velocity*. The main clusters are visualized by the four images depicted in figure 3.23. Clicking on these images opens a popover containing the respective gathering of patterns. The four sub-menus are shown in figure 3.24. Selecting one of the pattern images navigates to an event definition screen, visible in figure 3.26. By enabling the *Select Object*-Button, entering a label into the textfield below and clicking on an object in the screen adds the respective object for the event detection. Likewise the area is selected, if the search includes a defined region. An object can also be deleted by selecting it in the combo box and pressing the *Delete Object*-Button. Further options can be adjusted by the slider and the check boxes, depending on the selected pattern, as described above. Clicking on *Search* enables the search and depicts the corresponding results.

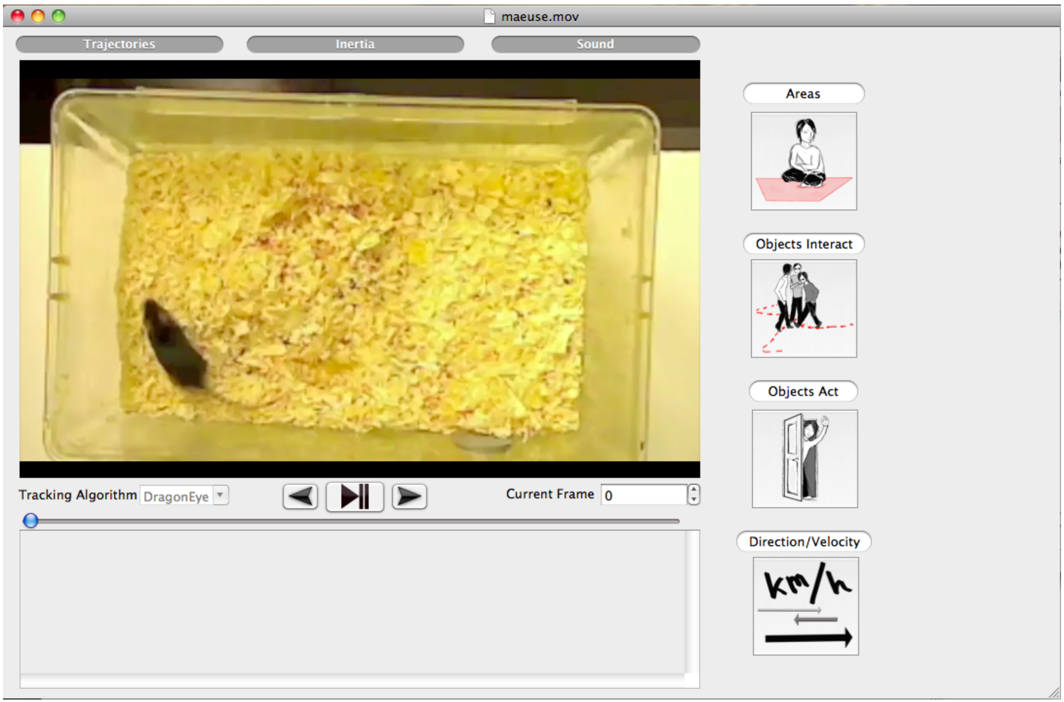

Figure 3.23: User Interface of event detection software: Main Menu. Video by B. Zipser.

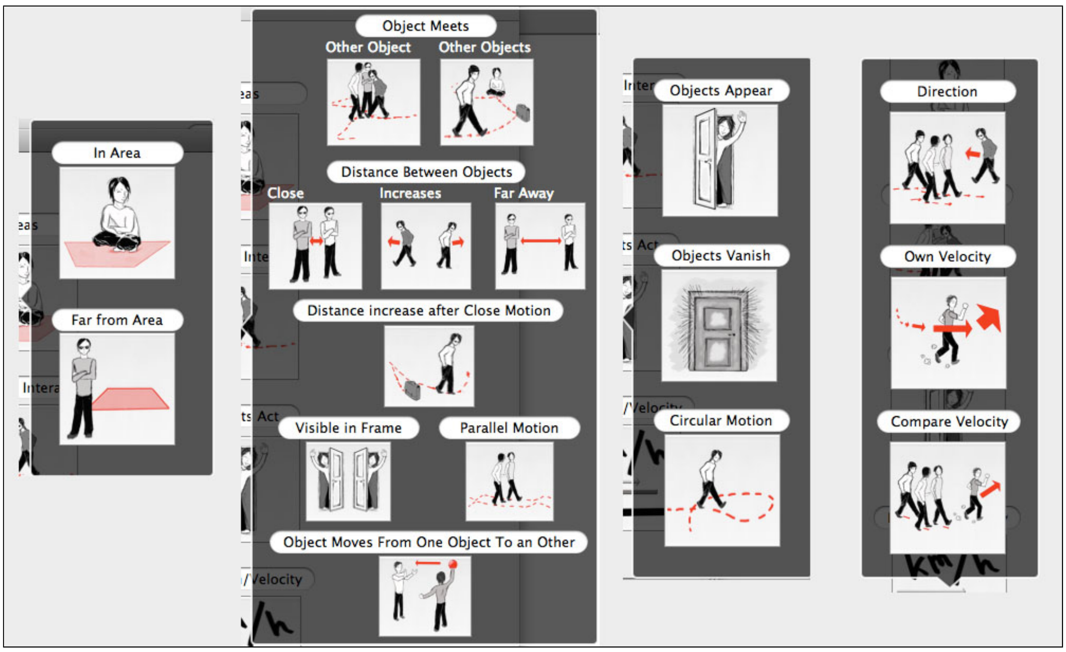

Figure 3.24: Submenus for pattern selection of the event detection software.

Figure 3.25: Submenu for pattern selection of the event detection software overview. Video by B. Zipser.

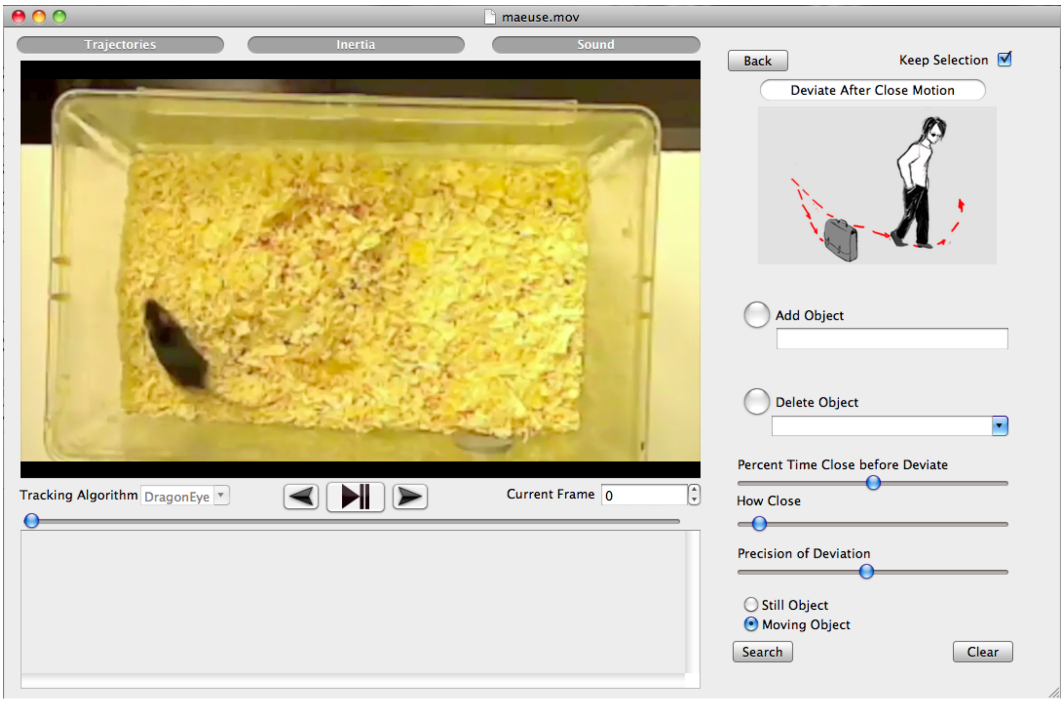

Figure 3.26: Selection criteria after a pattern has been chosen. Video by B. Zipser.

Chapter 4

Evaluation

DRAGON's event detection extension was tested in performance and usability. To test performance I accomplished precision-recall tests with a total of 26 videos, which are presented in section 4.1. Usability was checked in a user-study with 12 subjects. Experiments and results are demonstrated in section 4.2.

Evaluation

4.0.4 Precision Recall Tests

To approximate precision recall values I tested the seventeen algorithms on a minimum of three videos each from 26 recorded videos. The recordings where constituted of 19 real video tapings recorded by different cameras and 7 synthetic videos composed on the PC. Each video contains sequences in which the patterns occur. Precision was measured by calculating the percentage of frames from the correct sequences, which were identified properly by the pattern-recognizer. Recall values were computed by determining the portion of false positives, which where detected by the pattern recognizer compared to the total number of frames. Tests where tracking was too noisy were not considered in the results. The test material shows various scenes, which are described below. Video data was derived from different sources. Besides the synthetic material, I taped everyday-life situations, e.g. a woman cooking. I also used video material Karrer et al. created for the

Precision-Recall-Tests

DRAGON-studies and data, which was published by the PETS-workshop [PET]. Following up each video-sequence is presented. The total number of frames is specified in brackets behind each movie title.

Test-Videos

Real Videos:

1. figuresMeet.mov (684 f), figures2.mov (132 f), figures3.mov (269 f), figuresParallel.mov (345 f): Movies showing ludo pawns performing different movements.
2. Volleyball.Dragon (350 f), Volleyball_4.mov (358 f): Sequence of a volleyball match.
3. Kitchen.Dragon (1843 f): Shows a woman working in the kitchen.
4. ParallelIndependant.Dragon (796 f), Parallel1.Dragon (722 f), Parallel2.Dragon (512 f): Two pedestrians walking along a street in different constellations.
5. Troll1.mov (454 f), Troll2.mov (239 f), Troll3.mov (251 f): A toy troll moving circular. The movement is adapted from the motion of mice, seen in video material I gained from Benjamin Zipser on behavioral research of mice.
6. Mayersche.Dragon (276 f): Shows the top view of a street, with passing cars, bicyclists and pedestrians.
7. Billiard2.Dragon (87 f): Sequence of a billiard party.
8. Band.Dragon (386 f): Shows a band-conveyor of a canteen.
9. Pedestrian.Dragon (257 f): A pedestrian walking out of a building.
10. Pyramide.mov (23 f): Wooden cubes, which are hit by a tennis ball.
11. Pets 2006:MulticameraPersonTracking.mov (1205 f): Top view of people moving in a rail-way-station hall.
12. 2D-PersonTracking.mov Pets 2009 (62 f): Shows pedestrians on a street.

13. Tracking in a Parking Lot (545 f): Shows cars and pedestrians walking or driving through a parking lot.

Synthetic Videos:

1. Passen.mov (120 f): A circle which is passed between two other circles.
2. Kreisvideo.mov (325 f): A circle performing a circular motion.
3. Kreisvideo8.mov (219 f): A circle performing an 8-like motion.
4. deviate.mov (99 f): Two circles moving away from each other, with imperfect trajectories.
5. parallel3.mov (156 f): Two circles moving nearly parallel.
6. differentDirection.mov (114 f): Shows four circles; one moves into the opposite direction compared to the other three.
7. lostluggagestill2.mov (161 f): Two circles which are close first. After a certain amount of frames one circle stays still and the other one moves away.

Results: Area Dependancies

The cluster *Area Dependancies* shows very precise results and gives no false positives. Findings are listed in table 4.1. Total recall values of 0% and precision of 100% derive from the fact that users define areas by themselves and trajectory points are checked against these areas.

Results: Area Dependencies

Results: Objects Act

Results: Objects Act

The cluster *Objects Act* also shows very precise results, described in table 4.2. The appearing and disappearing is already defined in the trajectories. A circle is recognized,

Pattern	**Video**	**Precision**	**Recall**	**Options**
In Area	Band	$\frac{107}{107} = 100\%$	$\frac{0}{386} = 0\%$	1 Object
		$\frac{0}{0} = 100\%$	$\frac{0}{772} = 0\%$	2 Objects, independent
		$\frac{40}{40} = 100\%$	$\frac{0}{386} = 0\%$	2 Objects, dependent
	Billiard2	$\frac{67}{67} = 100\%$	$\frac{0}{174} = 0\%$	2 Objects, independent
	Volleyball	$\frac{24}{24} = 100\%$	$\frac{0}{700} = 0\%$	2 Objects, independent
		$\frac{0}{0} = 100\%$	$\frac{0}{700} = 0\%$	2 Objects, independent
		$\frac{0}{0} = 100\%$	$\frac{0}{350} = 0\%$	2 Objects, dependent
Far from Area	Kitchen	$\frac{9}{9} = 100\%$	$\frac{0}{1843} = 0\%$	1 Object
	Band	$\frac{183}{183} = 100\%$	$\frac{0}{368} = 0\%$	1 Object
		$\frac{295}{295} = 100\%$	$\frac{0}{772} = 0\%$	2 Object, independant
		$\frac{112}{112} = 100\%$	$\frac{0}{368} = 0\%$	2 Objects, dependent
	Pedestrian	$\frac{0}{0} = 100\%$	$\frac{0}{257} = 0\%$	1 Object
	ParallelIndependent	$\frac{0}{0} = 100\%$	$\frac{0}{796} = 0\%$	2 Objects, dependent

Table 4.1: Precision-Recall Results of Cluster *Area Dependencies*

when at least one starting point of a circle is returned. Due to the loose threshold settings in the pattern *Object Trajectory forms a Circle* all tested circle-like structures are recognized. Images of the respective trajectories are depicted in appendix A.

Pattern	**Video**	**Precision**	**Recall**	**Options**
Objects Disappear	Band	$\frac{3}{3} = 100\%$	$\frac{0}{386} = 0\%$	3 Objects
	Billiard2	$\frac{2}{2} = 100\%$	$\frac{0}{87} = 0\%$	2 Objects
	Volleyball_4	$\frac{1}{1} = 100\%$	$\frac{0}{358} = 0\%$	1 Objects
Objects Appear	Volleyball_4	$\frac{1}{1} = 100\%$	$\frac{0}{358} = 0\%$	1 Objects
	Billiard2	$\frac{3}{3} = 100\%$	$\frac{0}{87} = 0\%$	3 Objects
	Kitchen	$\frac{1}{1} = 100\%$	$\frac{0}{1843} = 0\%$	1 Object
Circular Motion	Kreisvideo	$\frac{1circle}{1circle} = 100\%$	$\frac{0}{325} = 0\%$	1 Object
	Kreisvideo8	$\frac{2circle}{2circle} = 100\%$	$\frac{0}{219} = 0\%$	1 Object
	Troll1	$\frac{1circle}{1circle} = 100\%$	$\frac{0}{454} = 0\%$	1 Object
	Troll2	$\frac{1circle}{1circle} = 100\%$	$\frac{0}{239} = 0\%$	1 Object
	Troll3	$\frac{1circle}{1circle} = 100\%$	$\frac{0}{251} = 0\%$	1 Object

Table 4.2: Precision-Recall Results of Cluster *Objects Act*

Pattern	Video	Precision	Recall	Options
Objects Meet	Mayersche1	$\frac{196}{206} = 95.14\%$	$\frac{0}{552} = 0\%$	2 Objects, independent
		$\frac{176}{176} = 100\%$	$\frac{0}{552} = 0\%$	2 Objects, independent
		$\frac{0}{0} = 100\%$	$\frac{0}{276} = 0\%$	2 Objects, dependent
	Billiard2	$\frac{3}{3} = 100\%$	$\frac{0}{87} = 0\%$	2 Objects, dependent
	ParallelIndependent	$\frac{541}{541} = 100\%$	$\frac{0}{1592} = 0\%$	2 Objects, independent
	figuresMeet	$\frac{85}{85} = 100\%$	$\frac{0}{684} = 0\%$	2 Objects, dependent
Meet Several	figures3	$\frac{51}{71} = 71.83\%$	$\frac{0}{269} = 0\%$	3 Objects
	passen	$\frac{48}{49} = 97.95\%$	$\frac{0}{120} = 0\%$	3 Objects
	Mayersche1	$\frac{83}{83} = 100\%$	$\frac{0}{276} = 0\%$	3 Objects
Objects Close	figures2	$\frac{41}{41} = 100\%$	$\frac{0}{132} = 0\%$	2 Objects
	Mayersche1	$\frac{22}{22} = 100\%$	$\frac{0}{276} = 0\%$	3 Objects
	Billiard2	$\frac{3}{3} = 100\%$	$\frac{0}{87} = 0\%$	2 Objects
Objects Far	Billiard2	$\frac{40}{40} = 100\%$	$\frac{0}{87} = 0\%$	2 Objects
	Band	$\frac{386}{386} = 100\%$	$\frac{0}{386} = 0\%$	2 Objects
		$\frac{237}{237} = 100\%$	$\frac{0}{386} = 0\%$	2 Objects
	figures3	$\frac{77}{77} = 100\%$	$\frac{0}{269} = 0\%$	3 Objects
Objects Deviate	figures3	$\frac{22}{55} = 40\%$	$\frac{0}{269} = 0\%$	
	deviate	$\frac{96}{96} = 100\%$	$\frac{3}{99} = 3.03\%$	
	mayersche	$\frac{115}{117} = 98.29\%$	$\frac{0}{276} = 0\%$	
	billiard	$\frac{42}{42} = 98.29\%$	$\frac{2}{87} = 2.3\%$	
Same Frame	figuresMeet	$\frac{135}{135} = 100\%$	$\frac{0}{684} = 0\%$	3 Objects
	Billiard2	$\frac{85}{85} = 100\%$	$\frac{0}{87} = 0\%$	2 Objects
	Volleyball_4	$\frac{50}{50} = 100\%$	$\frac{0}{358} = 0\%$	2 Objects
Parallel Motion	Parallel1	$\frac{187}{187} = 100\%$	$\frac{0}{722} = 0\%$	2 Objects
	Parallel2	$\frac{78}{143} = 54.54\%$	$\frac{0}{512} = 0\%$	2 Objects
	Parallel3	$\frac{28}{52} = 53.84\%$	$\frac{0}{156} = 0\%$	2 Objects
	ParallelFigures	$\frac{238}{249} = 95.58\%$	$\frac{0}{345} = 0\%$	2 Objects
Object Passed	passen	$\frac{6}{6} = 100\%$	$\frac{0}{120} = 0\%$	3 Objects, stability 0.11
	figures3	$\frac{1}{4} = 25\%$	$\frac{0}{269} = 0\%$	3 Objects, stability 0.57
	Billiard2	$\frac{2}{2} = 100\%$	$\frac{0}{87} = 0\%$	3 Objects, stability 0.5
Deviate after Close	TrackingParkingLot	$\frac{33}{33} = 100\%$	$\frac{0}{545} = 0\%$	2 Objects
	lostluggagestill2	$\frac{63}{63} = 100\%$	$\frac{0}{161} = 0\%$	2 Objects
	figuresParallel	$\frac{74}{74} = 100\%$	$\frac{22}{345} = 6.37\%$	2 Objects

Table 4.3: Precision-Recall Results of Cluster *Objects Interact*

Results: Objects Interact

Results: Objects Interact

Test results of the cluster *Objects Interact* are depicted in table 4.3. Recall tests of *Objects Meet* and *Object Meets Several other Objects* show an occurence of 0%. The precision values are less perfect: *Objects Meet* results in a precision of 99.19%; *Objects Meets Several other Objects*'s precision adds up to 89.92%. This derives from the problem that average object sizes are not estimated for this calculation, therefore the precision of this algorithm is limited. The patterns *Distance Between Objects: Close* and *Distance Between Objects: Far* return satisfying results, since the user is able to adjust minimum and respectively maximum distance. Absolute precision of *Distance Between Objects: Increases* accounts 84.57%. The outlying precision-value of figures3 occurs because one of the tracked objects does not move, which causes that linear regression does not return useful results. To improve this algorithm the test could be extended by checking if the objects move. In this case results are more sufficient, when only distance computation is performed. The measured total recall value accounts 1.33%. Variations in both values are caused by the setting of the stability index. To improve this value an analysis of the regression model needs to be performed in advance. Also the precision value of *An Object moves from one Object to an Other* accounts 75% and recall-tests of *Distance between two Objects increases after Close Motion* result in 2.12%, since these patterns are derived from the patterns described above. The pattern *Objects are visible in the same Frame* shows satisfying results. The *Objects have parallel Trajectories*-Precision result accounts 75.99%, since parallel moving objects do not necessarily create parallel trajectories. The maximal sicp-error is dependent on the intersection distance, by tracking object sizes this value could be improved likewise.

Results: Direction and Velocity

Results: Direction and Velocity

The analysis results of cluster *Direction and Velocity* are demonstrated in table 4.4. Total precision of the pattern *Objects move into opposing directions compared to the average Object Motion* adds up to 90.5%. Missed frames can result

Pattern	Video	Precision	Recall	Options
Different Directions	Mayersche1	$\frac{57}{69} = 83.6\%$	$\frac{0}{828} = 0\%$	3 Objects
	differentDirections	$\frac{92}{96} = 95.8\%$	$\frac{0}{456} = 0\%$	4 Objects
	MultiCameraPersonTracking	$\frac{84}{84} = 100\%$	$\frac{0}{3615} = 0\%$	3 Objects
	2D-PersonTracking	$\frac{19}{23} = 82.6\%$	$\frac{0}{182} = 0\%$	3 Objects
VelocityOthers	Mayersche1	$\frac{32}{32} = 100\%$	$\frac{0}{828} = 0\%$	3 Objects
	Pyramide	$\frac{8}{8} = 100\%$	$\frac{0}{69} = 0\%$	3 Objects
	TrackingParkingLot	$\frac{25}{37} = 67,57\%$	$\frac{0}{1635} = 0\%$	3 Objects
	differentDirections	$\frac{4}{4} = 100\%$	$\frac{1}{114} = 0.87\%$	3 Objects
VelocitySelf	Mayersche1	$\frac{121}{121} = 100\%$	$\frac{0}{276} = 0\%$	1 Object
	Band	$\frac{284}{284} = 100\%$	$\frac{55}{368} = 14.95\%$	1 Object
	DifferentDirection	$\frac{76}{76} = 100\%$	$\frac{0}{114} = 0\%$	1 Object

Table 4.4: Precision-Recall Results of Cluster *Direction and Velocity*

from noisy data, when the trajectories, do not follow the object's path correctly, or when a part of the object is tracked, which performs a separate motion, e.g. tracking an arm of a person while she is walking. The pattern *Velocity differs from average Velocity of all Trajectories* holds a total precision of 91.89%. Here frames can be omitted because of strong differences compared to still objects, if there are only few objects traced, as it is the case. Total recall of the pattern *Velocity differs from own average Velocity* accounts 4.98%. Since average values are used in this pattern the results are only meaningful if the object is traced over many frames, or if the difference between frames is less significant. To improve these patterns the comparison with the average value should be extended by further tests, where frames with too high deviation are discarded. All other analysis-outcomes were satisfying.

4.0.5 User Study

User Study

To test usability of the event detection extension of *DRAGON* I performed a user study with 12 subjects. Main goals of this first user study was to find out if the event detection system accelerates search on videos and which UI-elements should be improved, changed, or added to the system. Acceleration was measured by performing a paired students t-test on the null-hypothesis and measuring significance of the results. Users evaluated the system by answering the questions of the System Usability Scale.

Test Set-Up

Hypothisis

I hypothesized that search tasks in videos would be solved significantly faster using the developed event detection system than searching with a typical timeline-slider software. To test significance I designed a user study in which task completion time of both systems was measured and compared. 12 probands, 9 male and 3 female, at the age of 20 to 27 participated in the user study. Each regularly uses computers and had low to medium experience in video processing and analysis. All subjects were familiar with standard timeline sliders. Every user had to perform three search tasks in videos on two different systems. The first system was the event detection software, where user interface and functions were reduced, to elements the user needed for this study. Users were able to select one of the three search patterns: *Objects in Area, Objects Far from Area*, and *Object Meets several other Objects*. The interface is depicted in figures 4.1 and 4.2. Tracking of relevant objects was performed beforehand and functions for adding and deleting objects were disabled. First users were provided with information on the three search patterns. When the user felt familiar with the system, after a period of vocational adjustment, I presented the second system: a simple timeline-slider system, as shown in figure 4.4, which the user could test before solving search tasks as well. One after another I presented the search tasks and measured completion time. The task was to find specified situations on tapings of ludo matches:

Test Set-Up

Tasks

Task 1:
Find all situations in which one of the blue pawns enters the start-area.

Task 2:
Find all situations in which one of the blue pawns leaves the start-area.

Task 3:

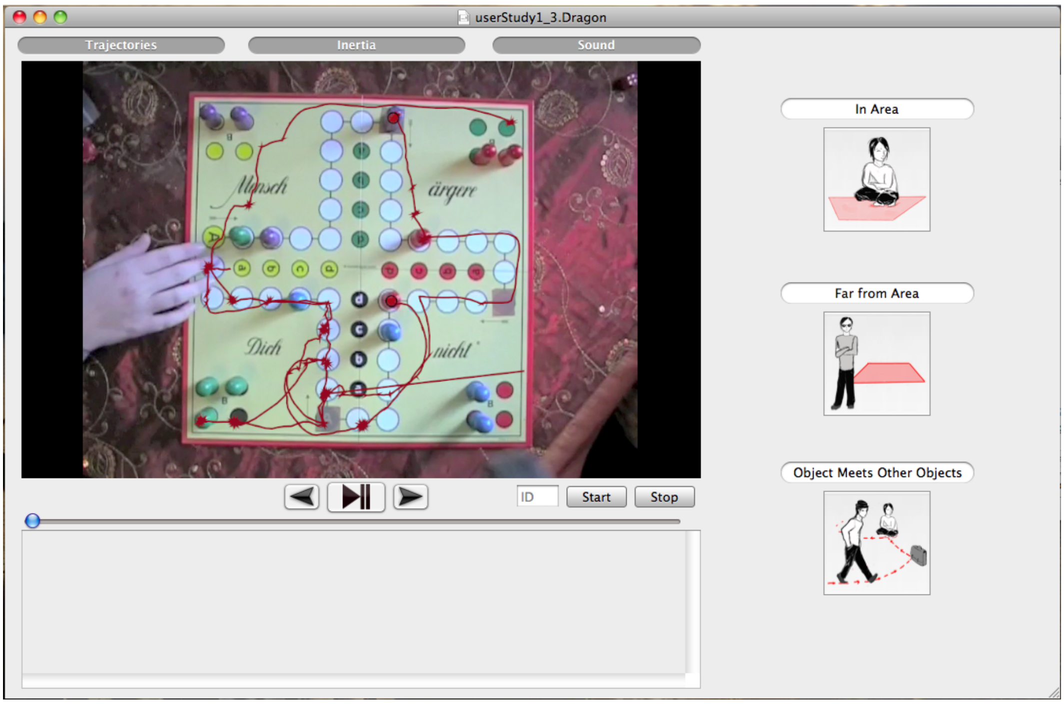

Figure 4.1: Reduced test interface of DRAGON for user-study containing only patterns *Objects in Area*, *Objects Far from Area*, and *Object Meets several other Objects*

Find all situations where purple meets red1, red2 ore green. (see figure 4.3)

All six combinations in ordering these tasks were tested. Each of the three tasks was performed on an individual ludo-video. First the users had to perform one of these tasks on the event detection software. After that they solved the same problem with the timeline slider system. The user was asked to click on start when she stated that she understood the given task and press stop when she believed to have found all relevant situations. Time was measured in this interval.

Significance and Acceleration Results

Individual Results

In 31 out of 36 cases users performed tasks faster using the event detection software. Individual completion times for each task can be viewed in table 4.5 to table 4.7.

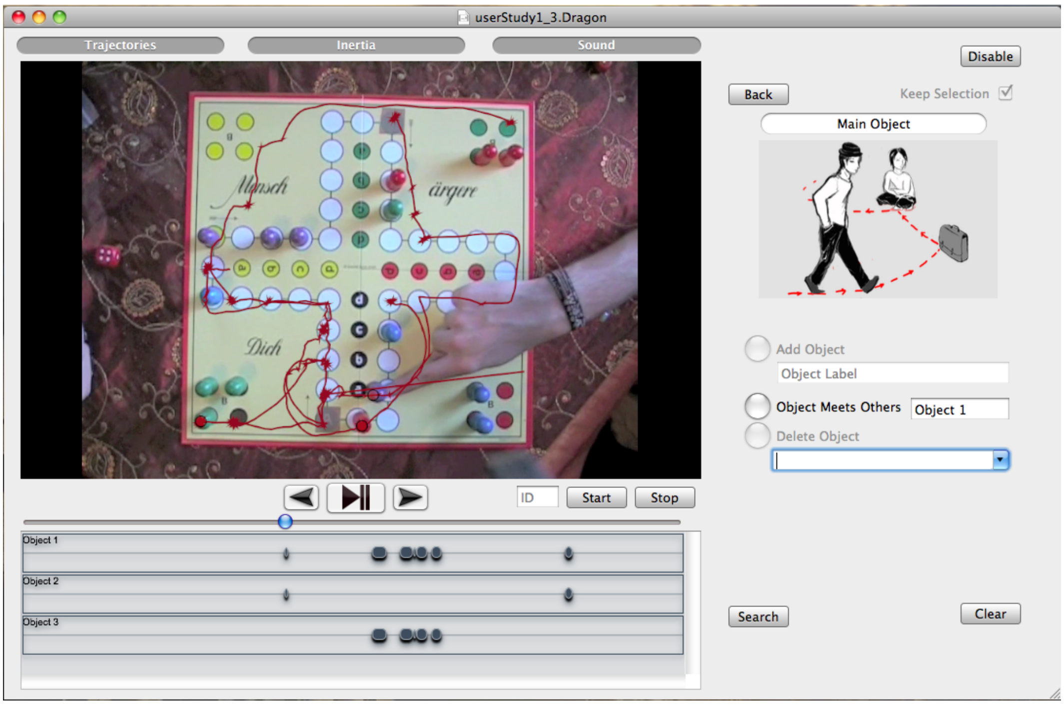

Figure 4.2: Submenu and results of reduced test interface of DRAGON for user-study.

ID	Dragon	Slider	Ratio: Dragon/Slider	Difference
1	90 s	135 s	0.6667	45 s
2	22 s	64 s	0.3438	42 s
3	23 s	34 s	0.6765	11 s
4	17 s	23 s	0.7391	6 s
5	24.8 s	45.5 s	0.5451	20.7 s
6	49.6 s	58.9 s	0.8421	9.3 s
7	51.8 s	40 s	1.295	-11.8 s
8	19.2 s	61.7 s	0.3112	42.5 s
9	22.4 s	29.7 s	0.7542	7.3 s
10	16 s	42 s	0.381	26 s
11	35 s	46.5 s	0.7527	11.5 s
12	16.8 s	42.3 s	0.3972	25.5 s

Table 4.5: User-Study Results of Task 1 (Enter Surface)

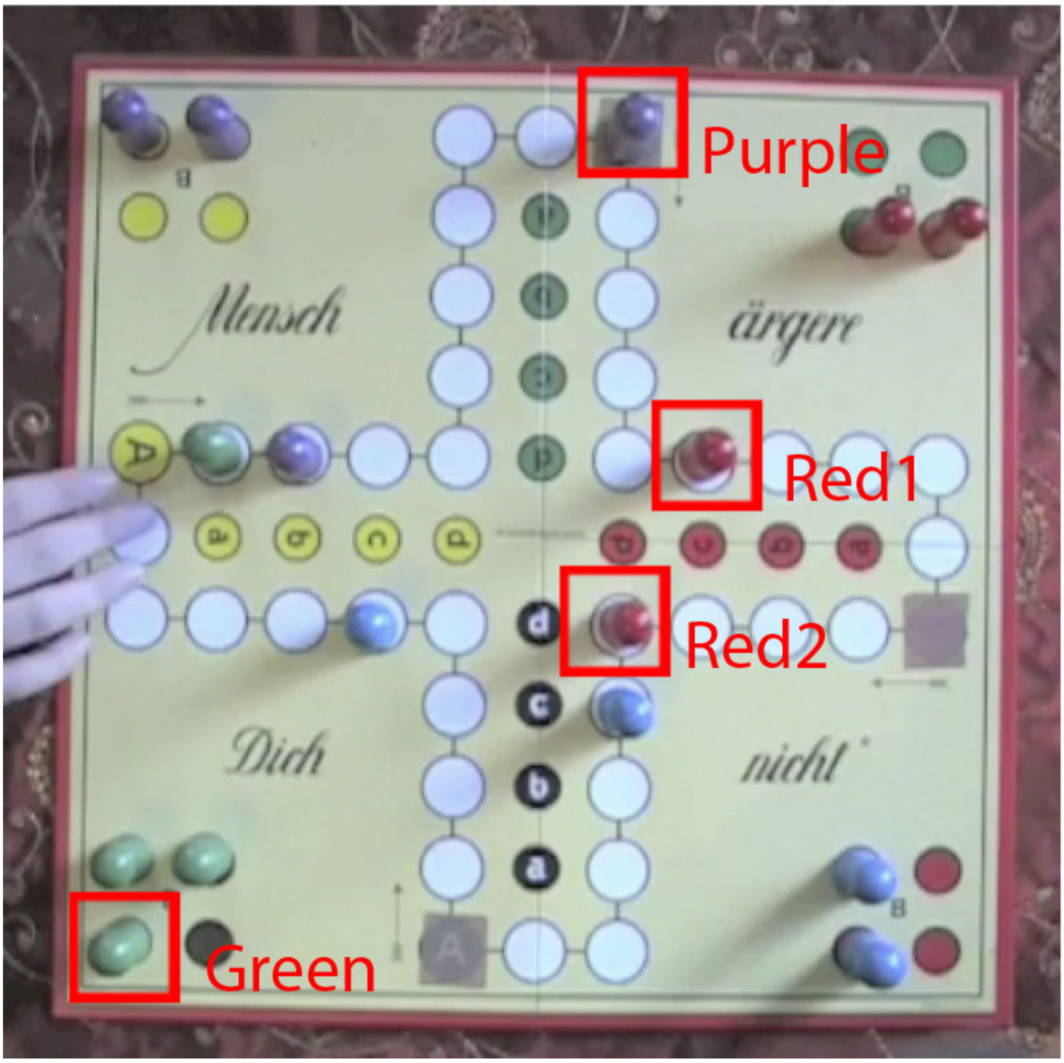

Figure 4.3: Start positions of involved pawns in task 3.

Significance: Paired Students t-test

Assuming the null-hypothesis, I performed a paired students t-test on the three data sets to find out if the performance time of the systems differs significantly [tTe]. Table 4.8 shows the results of the paired student's t-test. p denotes the probability that the null-hypothesis holds true. In all cases the difference between the data sets was significant ($p \leq 0.01$). Mean completion times and mean difference are listed as well. Obviously users perform faster on the event-detection system in average, even though event-locations were already presented in the first task. Since the values hold strong variance, the mean ratios *comple-*

Figure 4.4: Test-Interface of timeline-slider system.

ID	Dragon	Slider	Ratio: Dragon/Slider	Difference
1	42.9 s	74 s	0.5797	31.1 s
2	36 s	69 s	0.5217	33 s
3	55 s	119 s	0.4622	64 s
4	28.4 s	37 s	0.7676	8.6 s
5	25.7 s	58.5 s	0.4393	32.8 s
6	56.8 s	42.6 s	1.3333	-14.2 s
7	38.2 s	40.8 s	0.9363	2.6 s
8	31.3 s	52 s	0.6019	20.7 s
9	34.1 s	27.9 s	1,2222	-6.2 s
10	24.7 s	44 s	0.5614	19.3 s
11	32 s	48.2 s	0.6639	16.2 s
12	24.5 s	39.6 s	0.6187	15.1 s

Table 4.6: User-Study Results of Task 2 (Leave Surface)

ID	Dragon	Slider	Ratio: Dragon/Slider	Difference
1	126 s	139 s	0.9065	13 s
2	34 s	136 s	0.25	102 s
3	58 s	79 s	0.7342	21 s
4	37 s	67 s	0.5522	30 s
5	44.3 s	63.03 s	0.7028	18.73 s
6	67 s	141 s	0.4752	74 s
7	60 s	58.2 s	1.0309	-1.8 s
8	49.4 s	100.3 s	0.4925	50.9 s
9	58 s	72.2 s	0.8033	14.2 s
10	44.5 s	63.7 s	0.6986	19.2 s
11	108 s	91 s	1.1868	-17 s
12	23.5 s	98.3 s	0.2391	74.8 s

Table 4.7: User-Study Results of Task 3 (Meet)

tion time dragon/ completion time slider are itemized. Using the event detection software users required averaged only 62%-72% of the completion time that was needed for the slider system. In three tests users missed an event using the

Video	Length	Mean Time Dragon	Mean Time Slider	Mean Ratio	Mean Diff.	p
1	140 s	32,3 s	51,8833 s	0,64203	19,5833 s	0,002
2	152 s	35,8 s	54,3833 s	0,72569	18,5833 s	0,01
3	134 s	59,1417 s	92,3942 s	0,6209	30,6946 s	0,007

Table 4.8: Average Values and Result of Paired Student's t-test

Failures

timeline-slider system and often false positives were identified, because users lost the overview of the involved pawn locations, since object trajectories were not visible using the timeline-slider software. Users performed faster using the timeline-slider system in five cases. There were three reasons for these occurrences. In some cases users were unsure defining areas, multiple deletion and re-selection of the area delayed completion. Second some users were not sure if they could trust the results of the system. They found the results quite fast, but kept stepping forwards and backwards until feeling confident completing the task. A third reason was that the event detection system was tested first in every task. This way one of the users thinking he could remind himself of the event-locations only skipped to two results and left out the third.

Faster on Slider

SUS Results

The system usability scale contains ten questions giving an overall view of subjective assessments of usability (Brooke). Participants can answer questions in form of likert charts on 5 point scales. Results are presented in table 4.9. Table 4.10 shows the respective evaluation. According to

ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
1	4	1	5	2	5	1	5	1	5	1
2	4	1	5	1	5	1	5	1	5	1
3	5	2	5	1	5	1	4	1	5	2
4	2	1	5	1	5	1	4	2	5	1
5	2	1	5	1	4	1	4	1	3	2
6	4	2	4	1	5	1	4	2	5	1
7	5	2	4	1	5	1	4	1	5	1
8	3	4	2	4	4	1	4	1	5	3
9	4	2	4	1	4	1	4	2	4	3
10	2	4	3	3	2	1	4	2	4	3
11	2	2	4	1	4	2	4	2	4	1
12	4	1	5	1	4	1	5	1	5	1

Table 4.9: Results System Usability Scale

Aaron Bangor and Miller, products with scorings in the high 70's to upper 80's have acceptable SUS-scores, but are not truly superior and can thus be improved. The event detection system is rated with a SUS-Scale of 82. Lewis et. al describe how the SUS-Scale can be divided into usability and learnability factors (Lewis and Sauro). Both scales are located at 82-86. One main critique of the users was that they were not sure if they could trust the event detection system. An animation which visualizes the transition of the events when stepping through the results would be beneficial. Furthermore it is not quite clear in which area of the result frame the events occur. This should be clarified either by assigning different colors to the object markers or by visually emphasizing the event-location in the frame rectangle. Furthermore users where confused by area selection, because no selection rectangle is shown while performing area definition. Some adjustments in the choice of the UI-elements should be done, e.g. users had the opinion that they had to click to many buttons before the actual search

User's Impressions

ID	SUS Score	Usability	Learnability
1	92,5	93,75	87,5
2	97,5	96,875	100
3	92,5	93,75	87,5
4	87,5	84,375	100
5	80	78,125	87,5
6	87,5	84,375	100
7	92,5	90,625	100
8	62,5	68,75	37,5
9	77,5	78,125	75
10	55	56,25	50
11	75	68,75	100
12	95	93,75	100
Mean	82,92	82,29	85,42
Standard Deviation	13,39	12,66	21,21

Table 4.10: Evaluation System Usability Scale

was performed. This costs time and should be adjusted. The search button could for example be replaced by an automatic search after all search constraints are defined. Area selection could be selected automatically when entering the pattern. One has to keep in mind that the users were mere provided with a reduced interface. Furthermore object selection was already accomplished. The performed study was basically focused on acceleration time. Thus before changing buttons a usability study in which users need to accomplish all steps should be arranged. Most users stated that they felt confident using the system and would prefer to use the event detection software over the traditional timeline-slider to execute search tasks. Especially when the event location in time is unknown most users stated that they would strongly dislike searching through the video with the timeline slider. Users also stated that button icons made it easier to select the respective pattern. Often frame stepping and timeline-sliding were used in combination to understand the event detection results, thus browsing in different accuracies was a useful feature. According to these results the event detection software must be designed more trustful. Users already prefer to use the event detection system over the timeline slider. Thus the user's impression, that she has found a correct event, needs to be tightened, first by giving her a feeling for the semantical location of the returned frame by visualizing transitions and second by indicating the spatial in frame location of the result.

Chapter 5

Summary and future work

In the final chapter I present an overview of this thesis and an outlook on the research that should be performed in the future. This contains improvements in pre calculation of tracking data as well as advancements that result from precision tests and the user study.

5.1 Summary and contributions

Acceleration of Search Tasks

This thesis presents a software, which accelerates search tasks on videos by automatically detecting predefined events in scenes. In the area of video processing and analysis I filtered out five areas in which researchers and editors spend plenty of time watching videos to find important scenes. These areas are video ethnography, sports analysis, behavioral research, video editing, and visual surveillance or forensics. As presented in chapter 2, research on event detection has already been performed for some of the described areas.

Clustering and Pattern-Design

To create an event detection software which is appropriate to find a great amount of scenes in all these application areas, I gathered important events and scenes from these fields by questioning involved people, reviewing related work on the field, and reading up contributions released by representatives of the respective application area. Based on this analysis I clustered the gathered events by comparing trajectory patterns, that occur in the corresponding scenes, to 17 event detectors. For pattern recognition on an input-video I describe 17 algorithms which are based on predefined object trajectories and rectangular areas, with which the described scenes can be found. Precision-recall tests show that results of event detection are reliable in most cases. To create a user interface that is well known or beneficial for the users I gathered information on software, which is currently in use, in some of the described application areas. Based on the findings I designed the user interface described in section 3.3.

User-Interface

User Study

To test the user interface and the acceleration of search task completion I designed a user study in which the event detection system was tested against a standard timeline-slider video player. Users were provided with search tasks on each video system. After completing a task on the event detection system they were asked to perform the same search with the slider system. In each test task completion time was measured. Results were compared by performing a paired students t-test under the assumption of the null-hypothesis. It turned out that task completion time of the two systems differed significantly. In average users only needed 62%-72% of the time for task completion with event detection compared to the slider-search. SUS-Results (average score of 82) and user comments show, that the system is easy to use. Furthermore users stated that they preferred using the event detection software over the slider-system.

Result: Acceleration of Search Tasks

Contributions

The main contributions of this thesis are:

- Clustering scenes from the application area to comparable trajectory patterns.
- Designing detection algorithms for the defined patterns.
- Designing a user interface for event detection.
- Testing precision and recall values of the implemented algorithms.

- Accomplishing a user study to demonstrate usability and acceleration of search tasks on the event detection system.

5.2 Future work

This thesis presents a first prototype for event detection software based on *DRAGON*. Extensions and improvements can be made in pre-calculation of tracked data, in user-interface-, and algorithm design. First I will present critique and adjustments proposed by the user. Additional tracking information which would benefit the reliability of the algorithms is described. Finally algorithmic improvements are presented, which are partially based on the additional tracking information and should be added and adjusted in the future.

5.2.1 User Interface

Trust Results

One main critique point, which arouse during the user study was that users were not sure if they were able to trust the results of the event-detection software. When using the system they wished for an animation, which makes the transition of the results comprehensible. One improvement would be blending in an arrow depicting the course of movements the respective objects perform. Furthermore it was hard to distinguish the selected objects, thus they should have different colored markers. Users found it hard to find the location of the event in the frame, thus the in frame location should be emphasized visually. Another mentioned critique point was that users felt that they had to push too many buttons before performing search, consequently the button-interaction and preselection of buttons should be adapted. A second user study should be performed where the whole system is in use. Here the users should be familiar with the system beforehand. When the system is designed more trustful and the event location is clarified, the users should be able to perform search even faster. Finally, a user study with users from the target

Emphasize Results

Reduce button presses

groups should be performed, so that the software can be adjusted according to their opinion.

5.2.2 Automatic Object Recognition and Computation of Object Sizes

Automatic Object Recognition

At the moment the event detection is based on flow-field tracking of pixels. The user needs to select the pixel she wants to track and then define search criteria. An automatic object recognition and detection of object sizes would be beneficial for several reasons. First, object selection time becomes negligible. Second, the user is able to perform event detection on all recognized objects. This is especially beneficial in areas like visual surveillance, when the surveillant needs to be alarmed when something unusual happens. She is not able to click on all objects beforehand. Furthermore, by detecting objects, the whole object can be tracked, not only a single pixel. Thus the tracking data is less noisy and the event-detection algorithms show more reliable results. Finally, by performing object recognition one is able to approximate object sizes. This information enables to improve algorithms like *Objects Meet*, by computing exact intersection distances.

Object Size Computation

5.2.3 3D-Information

Algorithmic Improvement by 3D-Information

Some event detection algorithms would benefit from 3D-location-information of the objects. The pattern *Objects have Parallel Trajectories* performs a mesh-registration accepting scalings from 0.3 to 3 due to perspective foreshortening. If the 3D position of the objects were known this scaling could be computed precisely or respectively the object trajectories could be adjusted to the 3D-information and nearly no scaling is affordable. Also other algorithms like deviation computations would benefit from 3D information, since it is possible to make them more precise based on this data.

5.2.4 Camera Motion Adjustment

At the moment the described algorithms can only be used on still frames. But especially in fields as sports analysis the camera is not still. Thus a camera motion adjustment should be added. Wittenhagen [2008] presented an approach for camera motion adjustment in *DRAGON*. A comparable camera adjustment should be added to the event detection extension.

Camera Motion

5.2.5 Algorithmic Improvement

Some of the demonstrated algorithms can still be improved. As described above *Objects Meet* and *Object Meets several other Objects* can be advanced by calculating object sizes. *Objects have Parallel Trajectories* and *Distance between Objects: Increases* can be enhanced by integrating 3D-information. Furthermore *Distance between Objects: Increases* should be improved regarding still objects. Some of the patterns like *Distance between two Objects increases after a close Motion* are based on the algorithms which need advancement. These should be tested again after the basic patterns are adjusted.

Improvements: Objects Interact

Velocity and Distance algorithms work on average values. When there are only few objects selected or when the video is short, very large values, which can also be caused by noisy data, distort the results. Thus further analysis should be performed where frames with too high deviation are discarded.

Improvements: Distance and Velocity

Appendix A

Images of Precision-Recall-Tests

Appendix A shows several screenshots as examples for experimental results taken during the precision-recall tests.

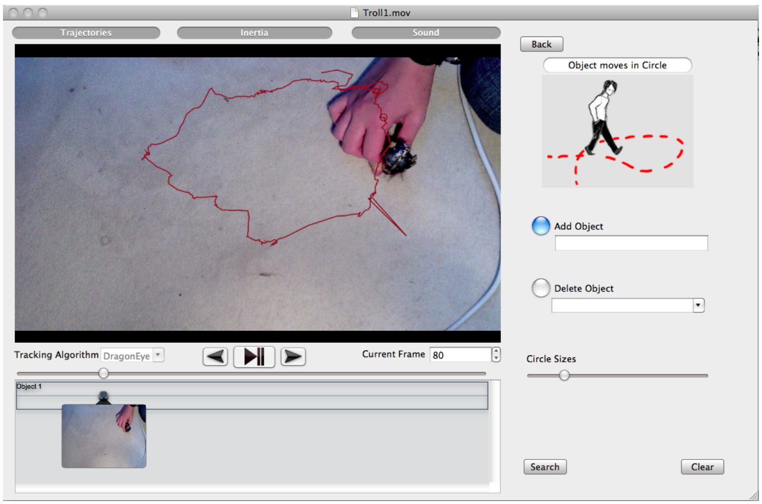

Figure A.1: Movie Troll1, where a troll-figure performs a circular motion.

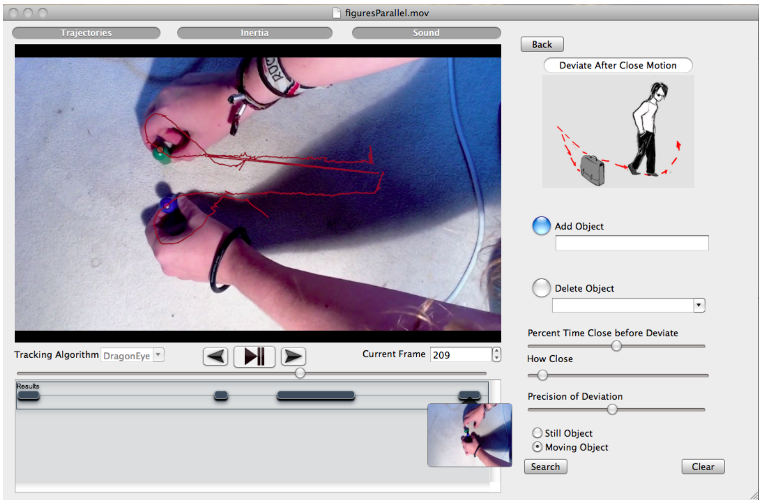

Figure A.2: Results of the pattern *Objects deviate after close motion* on the video figuresParallel.

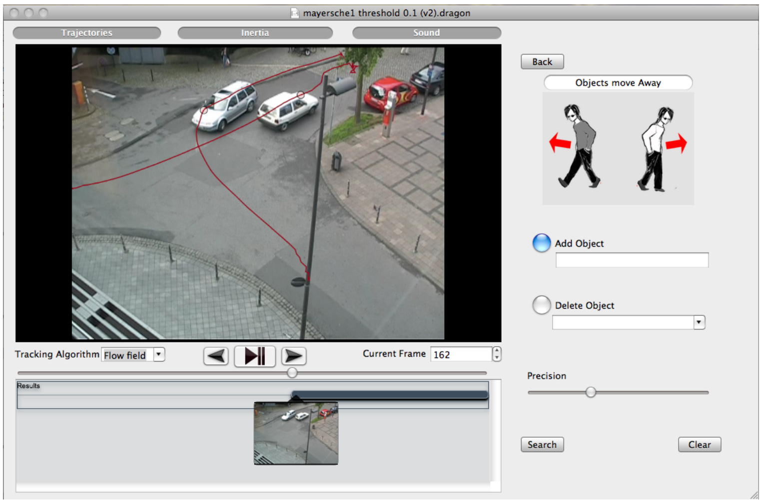

Figure A.3: Results of the pattern *Distance between Objects: Increases* on the video mayersche1.

Figure A.4: Results of the pattern *Objects far away from Area* on the video billiard2.

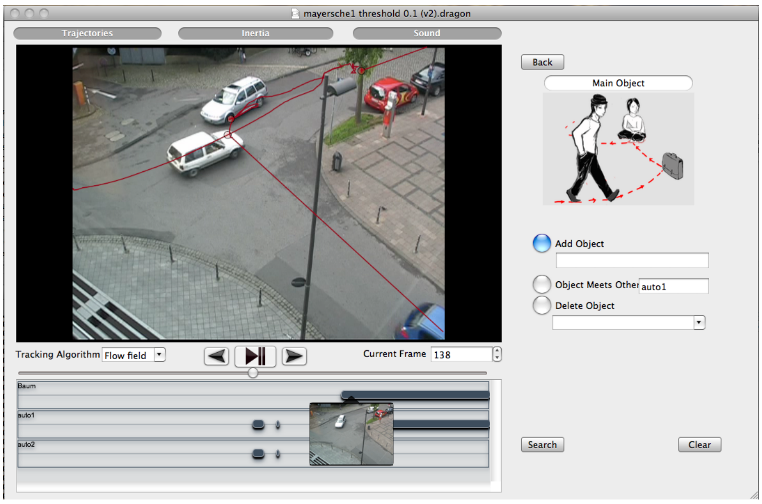

Figure A.5: Results of the pattern *Object meets several other objects* on the video mayersche1.

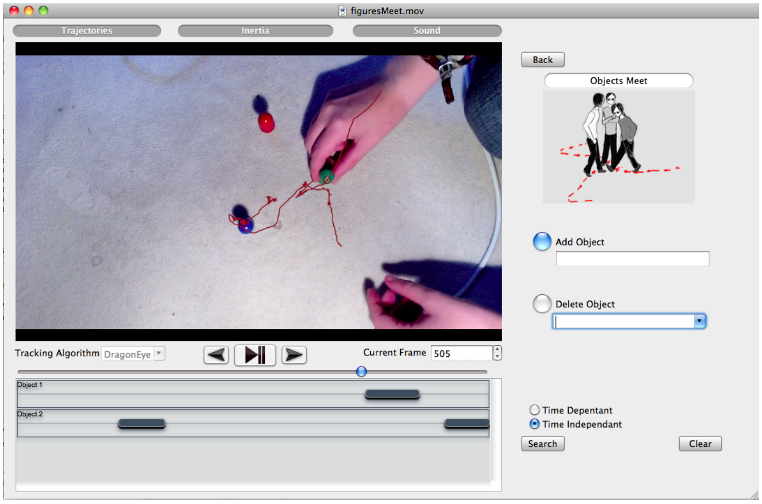

Figure A.6: Results of the pattern *Object meets several other objects* on the video figuresMeet.

Figure A.7: Results of the pattern *Object moves from one Object to an Other* on the video billiard2.

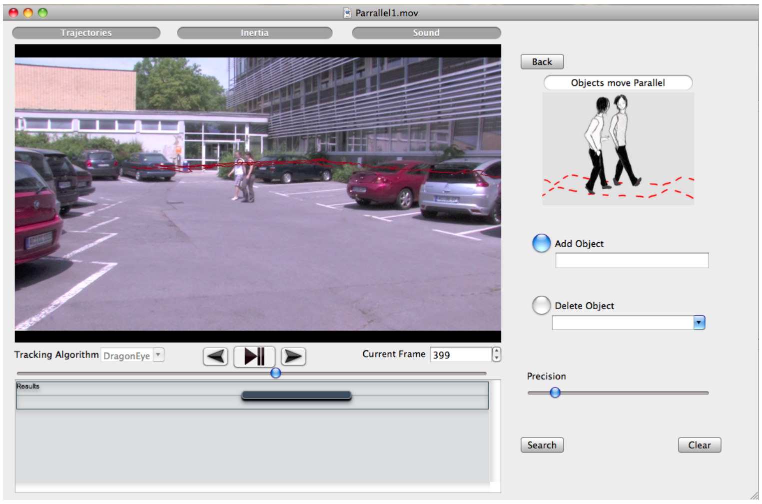

Figure A.8: Test-Results on parallel trajectories, where two objects are defined. (Video Parallel1)

Bibliography

Stoelting any-maze. www.anymaze.com.

Sports analytics gmbh. http://www.umwelt-campus.de.

Denver, Colorado Municipal Code. 1950.

Philips research experiencelab. www.research.philips.com.

Final cut, a. www.apple.com.

Finalcut, b. www.cancom.de.

Gesellschaft fuer innovative marktforschung. www.g-i-m.com.

Mangold interact. www.mangold-international.com.

Traffic surveillance system has capacity to catch offenders. *Jamaica National Building Society*, June 24, 2010. www.jnbs.com.

karthik energy technologies. http://karthikenergytech.tradeindia.com.

On the use of video content analysis in its: A review from academic to commercial applications.

Noldus observer, a. www.noldus.com.

Behavioral observation research group, b. sazcsufborg.org.

Reduction of episodes of seclusion and restraint in a psychiatric emergency service. *Psychiatric Services*, 2004 American Psychiatric Association. www.ps.psychiatryonline.org.

Pets workshop. www.cvg.rdg.ac.uk.

Adobe premiere. www.adobe.com.

Skyway security. www.skywaysecurity.com.

Schwerpunkt - umwelt-campus birkenfeld - mechanik ii. www.sports-analytics.de.

Paired student's t-test. mlsc.lboro.ac.uk/resources/statistics/Pairedttest.pdf.

Philip T. Kortum Aaron Bangor and James T. Miller. An empirical evaluation of the system usability scale.

Edwin Hutchins Adam S. Fouse, Nadir Weibel and James D. Hollan. Chronoviz: A system for supporting navigation of time-coded data. 2011a.

Edwin Hutchins Adam S. Fouse, Nadir Weibel and James D. Hollan. Supporting an integrated paper-digital workflow for observational research. 2011b.

A. Murat Tekalp Ahmet Ekin and Rajiv Mehrotra. Automatic soccer video analysis and summarization. In *IEEE Transactions on Image Processing*, volume Vol. 12, July 2003.

Don Kimber Andreas Girgensohn and Thea Turner Eleanor Rieffel. Dots: Support for effective video surveillance. In *MM'07, September 23–28, 2007, Augsburg, Bavaria, Germany*, 2007.

Edvin Babic. Ethnografische videodokumentationen in der pharma- und gesundheitsbranche. *Planungs-handbuch OTC 2010 Media Spectrum*, 2010.

Michael C. Frank Brandon C. Roy and Deb Roy. Exploring word learning in a high-density longitudinal corpus.

Daniel Brete. Computerpraktikum im gp ii: Lineare regression, 2004.

John Brooke. Sus - a quick and dirty usability scale.

William Chen and Shih-Fu Chang. Motion trajectory matching of video objects. In *Proc. SPIE*, volume Vol. 3972, p. 544-553, 2000.

Scott Mc Cloud. *Making Comics: Storytelling secrets of comics, manga and graphic novel*. Harper.

Erhard Cramer and Udo Kamps. *Grundlagen Der Wahrscheinlichkeitsrechnung Und Statistik*. Springer, 2008.

David Salesin Dan B. Goldman, Brian Curless and Steven M. Seitz. Schematic storyboarding for video visualization and editing. In *SIGGRAPH '06 ACM, ACM Transactions on Graphics (TOG)*, volume Volume 25 Issue 3, 2006.

Bo Wu Fengjun Lv, Xuefeng Song and Singh Ramakant Nevatia. Left luggage detection using bayesian inference. In *In PETS*, 2006.

Isaac Cohen Gerard Medioni and Francois Bremond. Event detection and analysis from video streams. In *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, volume VOL. 23, NO. 8, AUGUST 2001.

Abir Al Hajri Gregor Miller, Sidney Fels and Michael Ilich. Mediadiver: Viewing and annotating multi-view video. In *CHI 2011*, 2011.

Thomas Wolle Joachim Gudmundssona. Towards automated football analysis: Algorithms and datastructures. *Preprint submitted to MATHSPORT 2010*, 2010.

Jason Campbell Larry Huston, Rahul Sukthankar and Padmanabhan Pillai. Forensic video reconstruction. In *VSSN '04 Proceedings of the ACM 2nd International Workshop on Video Surveillance and Sensor Networks*, 2004.

James R. Lewis and Jeff Sauro. The factor structure of the system usability scale.

Alessandro Mecocci Lorenzo Favalli and Fulvio Moschetti. Object tracking for retrieval applications in mpeg-2. In *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, volume VOL. 10, NO. 3, APRIL 2000.

Wendy E. Mackay and Glorianna Davenport. Virtual video editing in interactive multimedia applications. In *Communications of the ACM*, volume Volume 32 Number 7, 1989.

Bennett Jackson Robert Bodor and Nikolaos Papanikolopoulos. Vision-based human tracking and activity recognition. In *Proc. of the 11th Mediterranean Conf. on Control and Automation*, pages 18–20. Kostrzewa Joseph, 2003.

Deb Roy. New horizons in the study of child language acquisition. *Invited keynote paper, Proceedings of Interspeech*, 2009.

Mario Borth Sandip Sar-Dessai and Leif Kobbelt. *Lecture Notes on Computer Graphic 1*. Rheinish-Westfaelische Technische Hochschule Aachen, Department for Computer Graphics and Multimedia.

Nicolas Saunier and Tarek Sayed. Automated road safety analysis using video data. In *Paper Submission to the 2007 TRB Annual Meeting*, 2007.

Shihui Ying Shaoyi Du, Nanning Zheng and Qubo You. An extension of the icp algorithm considering scale factor. In *2007. IEEE International Conference on Image Processing*, volume V - 193 - V - 196, ICIP 2007.

Monnique Thonnat Shobhit Saxena, Francois Bremond and Ruihua Ma. Crowd behavior recognition for video surveillance. In *Proceeding ACIVS '08 Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, 2008.

Nils T. Siebel and Stephen J. Maybank. The advisor visual surveillance system. In *in ECCV 2004 workshop Applications of Computer Vision (ACV*, volume V 193 - 196, 2004.

N. Papenberg T. Brox, A. Bruhn and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *European Conference on Computer Vision (ECCV)*, volume volume 3024 of LNCS, 25–36, 2004.

E. Lee T. Karrer, M. Weiss and J. Borchers. Dragon: a direct manipulation interface for frame-accurate in-scene video navigation. In *CHI '08: Proceeding of the Twenty-Sixth An- nual SIGCHI Conference on Human Factors in Computing Systems*, volume pages 247–250, New York, NY, USA, 2008.

Vasanth Tovinkere and Richard J. Qian. Detecting semantic events in soccer games: Towards a complete solution. In *2001 IEEE International Conference on Multimedia and Expo*, 2001.

Liang Wang Weiming Hu, Tieniu Tan and S. Maybankt. A survey on visual surveillance of object motion and behaviors. In *Transactions on Systems, Man, and Cybernetics, IEEE*, volume Part C: Applications and Reviews, 12 Juli 2004.

M. Wittenhagen. Dragoneye: Fast object tracking and camera motion estimation. Master's thesis, RWTH Aachen University, 2008.

Yasunobu Yanagisawa. Social behavior and mating system of the gobiid fish. In *Japanese Journal of Ichthyology*, volume VOL. 28, NO. 4, 1982.

Glossary

Adobe Premiere	Video Editing Software
Area	A rectangle defined by an offset (x, y), *width*, and *height*
Behavioral Research	All disciplines that explore the activities of and interactions among organisms in the natural world
Bounding Box	Smallest rectangle, which incloses a geometric object
Camera Motion Estimation	Computation of the camera motion to adjust trajectory coordinates to real world coordinates
ChronoViz	Video annotation software
Cinematic Features	Camera calibration, colors, and edge features in a frame
Cinematic Features	Cinematic Features
Cluster: Area-Object Interaction	Cluster containing all patterns in which objects interact with areas
Cluster: Direction and Velocty	Cluster containing all patterns in which the objects velocity or direction is peculiar
Cluster: Objects Act	Cluster containing all patterns of objects performing independent motion
Cluster: Objects Interact	Cluster containing all patterns where objects interact with each other
Dependency Value	Variable used in the clusters *Objects Interact* and *Area Dependencies*, which can only take the values T_1 and T_2
Distance Histogramm	Clustering of distances
DOTS	Event detection surveillance system

DRAGON	DRAGable Object Navigation: In scene navigation software
Event Detection Software	Software detecting predefined scenarios or sequences which fulfill given contraints
ExperienceLap	House filled with cameras created by the Phillips Research Group for ethnographic studies
Feature Vector	Vector describing attributes of an object
Final Cut	Video Editing Software
Forensics	Broad spectrum of sciences to answer questions of interest to a legal system. This may be in relation to a crime or a civil action.
GIM	Gesellschaft für Innovative Marktforschung: Company performing market research based on ethnographic studies
Interact	Video annotation Software
Linear Regression	Interpolation of a line through a point set
Lost-Luggage-Szenario	Scenario containing a person abandoning her luggage
Marching Corner Cutting	Algorithm to compute the surface of a polygon by itterativly cutting off corners.
Meen Value Filtering	Filters n values to a single one by computing their mean value
Mesh Registration	Assimilation of a polygon-mesh to an other mesh
Motion Arrows	Arrows depicting the motion of objects on a still image. Often used by story board artists
Null-Hypothesis	Assumption that there is no difference between the analyzed subjects
Object Tracking	Automated computation of object postions in a video
Object Trajectory	Data-structure holding position information for each frame at which the object is visible
Observer	Video annotation Software

Optical Flow Fields	Contain information on the most likely pixel locations in the succeeding and preceding frames
Orthogonal Projection	Projection onto a geometry by 90 degrees
Paired Student's t-Test	Significance test to check the difference of two subjects
Precision	Measures the correctness of an algorithm
Range	Data-structure containing an offset (location) and a length-value
Recall	Measures false positives of an algorithm
RFID	Radio-frequency identification: A technology that uses radio waves to transfer data from an electronic tag
SICP	Scale invariant closest point algorithm: Performs a scale invariant mesh registration
Sports Analysis	Analysis, advice, and commentary of sports matches
Stability Index	Value between 0 and 1. Denotes the correctness of a linear regression model
System Usability Scale	Ten questions on a five point scale to evaluate the user interface of a system
Trajectory Pattern	Trajectory constellation or shape fulfilling predefined constraints
Video Editing	The process of editing segments of motion video production footage, special effects, and sound recordings in the post-production process
Video Ethnography	The video recording of actors in their natural environment and context with the aim of eliciting insights, and applying that knowledge to process development, product development, new product development and product design
Visual Surveillance	The monitoring of the behavior, activities, or other changing information

Index

Typeset September 25, 2011