RWTH AACHEN UNIVERSITY

# DragonEye

## *Fast Object Tracking and Camera Motion Estimation*

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

by
Moritz Wittenhagen

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr.-Ing. Jens-Rainer Ohm

Registration date: April 30th, 2008
Submission date: Oct 30th, 2008

# Contents

# List of Figures

# List of Tables

# Abstract

Recently, multiple research groups proposed the concept of direct manipulation, which makes the task of navigation in videos directly related to content of the video. However, the developed systems are still too slow for a broad application of the technique. To increase the accessibility of direct manipulation a system has to fulfill certain requirements:

- Computation times have to be so small that the system can work in situation with time constraints.
- The system has to accommodate for the user's greater comprehension of a scene. This includes camera motion, fore- and background, and occlusions.

Based on the requirements, we developed and implemented the DRAGONEYE system. The tracker employs point tracking, with points provided by SIFT [Lowe, 2004], and color tracking with the CAMShift algorithm [Comaniciu et al., 2003]. A model of the tracked object is built and continuously updated to fit the current conditions. This model allows the detection and recovery from occlusions.
Secondly, the system estimates the effects of camera motion and represents them as a homography.
Tests of the system showed that the tracking results are comparable to the current tracking approaches, and the system outperforms the other algorithm in the case of occlusions. The tests also showed that improvements are still necessary in case of rotations around the y-axis and non-rigid movement.

# Überblick

Verschiedene Forschergruppen haben jüngst das Konzept der direkten Manipulation vorgestellt, das Navigation in Videos durch direkte Interaktion mit dem Inhalt erlaubt. Allerdings sind die entwickelten Systeme noch zu langsam für eine breite Anwendung dieser neuen Technik. Um die Zugänglichkeit der Systeme zu erhöhen müssen sie bestimmten Anforderungen genügen.

- Berechnungen des Systems müssen schnell erfolgen, um auch die Anwendung in Szenarien mit Zeitbeschränkung zu gewährleisten.
- Das System muss das größere Verständnis eines Benutzers beachten, und speziell Informationen über Kamerabewegung, Vor- und Hintergrund, sowie Verdeckungen in einer Szene haben.

Basierend auf diesen Bedingungen, haben wir das DRAGONEYE System entwickelt. Der Tracker verwendet Punkttracking, mit durch SIFT gefundenen Punkten, und Farbtracking mit dem CAMShift Algorithmus. Der Tracker erstellt ein Modell des getrackten Objektes und passt es in jedem Schritt den neuen Gegebenheiten an. Dieses Modell ist dann besonders nützlich um Verdeckungen zu erkennen und zu behandeln.
Tests zeigen, dass die Resultate unseres Systems mit den aktuellen Systemen vergleichbar sind und unser System die anderen Algorithmen im Fall einer Verdeckung übertrifft. Notwendige Verbesserungen betreffen hauptsächlich Rotation um die y Achse und Verformungen.

# Acknowledgements

First of all, I would like to thank Prof. Borchers for introducing me to the field of HCI and bringing a little light to the otherwise often dull and dry CS lectures. Special thanks goes to Thorsten Karrer and Malte Weiß for taking me on as a diploma thesis student. Then, I want to thank everyone who had to deal with me in the last 48 hours of this thesis and did not kill me. Especially, Robert Hochstrat and Nikolaus Koemm who did the main part of the proof reading. I want to thank Leonhard Lichtschlag, Nori Vontin and Mei-Fang Liau for great afternoons and nights at the chair. I want to thank my roomies Rafael Müller and Lukas Bernatzki for listening to my rants.
And finally, I want to thank my family for supporting me the last few months.
This is for everyone I forgot: Thank you all!

# Conventions

Throughout this thesis we use the following conventions.

*Text conventions*

Definitions of technical terms or short excursus are set off in colored boxes.

**EXCURSUS:**
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
*Excursus*

Source code and implementation symbols are written in typewriter-style text.

```
myClass
```

The whole thesis is written in American English.

# Chapter 1

# Introduction

Today, the production of videos has become more and more popular for average users over the last years. Systems like YouTube[1] allow for uncomplicated distribution of videos and encourage people to present their videos online. The process of video editing is made easier by products specifically designed for non professional users, for example Apple iMovie or Microsoft Movie Maker. These systems offer a low complexity and low entrance threshold. Professional systems, such as Apple Final Cut Pro, have a much higher threshold but give the user the possibility of more complex scene analysis; for example movement analysis which can be employed to estimate missing frames in a slow motion sequence. Video navigation is essential for the process of editing videos. Common approaches for navigation, like the timeline metaphor, require the user to abstract from the content of the video. If he is interested in an object's position, he has to understand how this is related to time, namely how long the object will take to get to the desired point. Afterwards, this has to be applied to the timeline, which represents time difference as a distance on a slider bar. Unfortunately, the distance on this timeline is dependent on the length of the video, forcing the user to estimate that distance as well. The same scene in a longer video is represented by less space on the timeline, making especially fine grained navigation impossible with the standard timeline slider. For the whole process, the user has to map

[1] http://www.YouTube.com

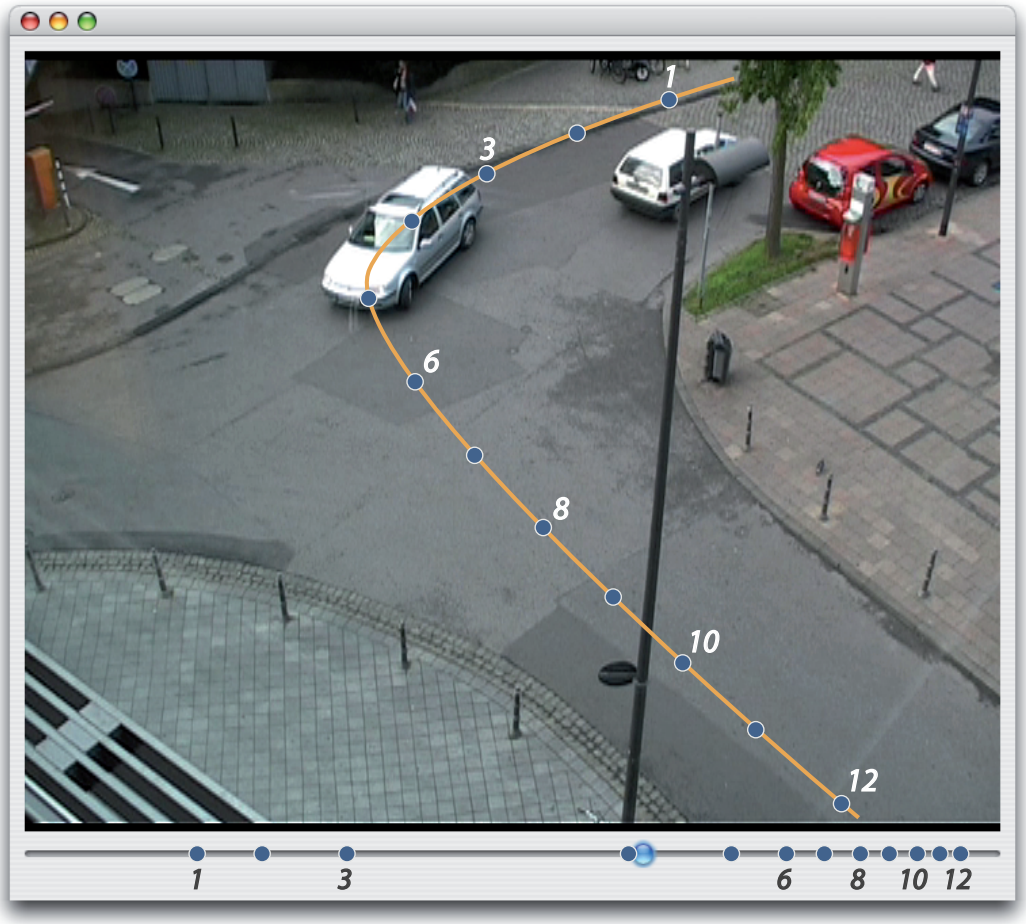


**Figure 1.1: Relation between object motion and timeline [Karrer et al., 2008]:** Since the car accelerates, the points on the timeline move closer together while the corresponding points on the path stay equidistant. The user has to be aware of the acceleration to be able to choose the correct point to drag the slider to

a distance to a time difference and he has to map this time difference back to another distance which is not related to the actual object movement anymore. Figure 1.1 visualizes this problem.

Direct manipulation

The more direct access to video provided by direct manipulation described by Karrer et al. [2008] is beneficial to beginners as well as professional users. First-time users have to learn less metaphors and do not need to gather experience on how they work, reducing the entrance threshold even further. Professional users can benefit from the reduc-

tion of cognitive steps necessary to complete a navigation task because the process can become more efficient and focussed on the task of editing. Several research groups have studied how to gain a more direct access to video content. Dragicevic et al. [2008] and Karrer et al. [2008] focussed on the problem of in-scene navigation, Kimber et al. [2007] presented a system for video surveillance and Goldman et al. [2006] developed a technique for representing scenes as a still frame that shows contained movement in the style of a storyboard. All these systems use the concept of direct manipulation at some point. Direct manipulation allows for video navigation based on video content. The video player is aware of how a pixel, or an object, moves and the user can then employ this information for navigation. The user clicks on the object he is interested in and can simply drag it to the position he wants it to be at. In this way, he does not have to abstract from the content. This makes the whole process metaphor free — there is no indirection in between an object moving because the video is playing and an object moving because the user drags it. Thus, direct manipulation reduces the cognitive load necessary for navigation and the user can focus more on his task, for example video editing. Consider the example in figure 1.1. If the user does not drag the slider but the car, the acceleration of the car does not affect the interaction. The user only needs to know where the car goes, which can be visualized very well by showing its path. These paths, or trajectories, have to be computed for visualization as well as the interaction. Current approaches for computing the trajectories for direct manipulation have the drawback of requiring a precomputation step. Furthermore, the algorithms are not designed to deal with some problems, such as occlusion, occurring in many videos. This warrants the development of a new, faster, algorithm, which will make the system more easily accessible to the general public.

## 1.1 Thesis Overview

This thesis deals with the development of a new tracking algorithm for DRAGON, the system originally developed by Karrer et al. [2008]. In chapter 2 — Related Work — we

present existing systems that employ direct manipulation and their respective application areas. Based on the properties described in related works, as well as some broader requirements, in chapter 3 — Requirements for Direct Manipulation — we define which properties an algorithm for direct manipulation must have. The chapter also shortly describes existing tracking algorithms and how well they fit the given requirements. With these requirements, we explain why the current approaches do not suffice for direct manipulation. In chapters 4 — Camera Motion Estimation — and 5 — Object Tracking — we describe the algorithms developed for this thesis. Chapter 6 — Evaluation — deals with the mathematical analysis of the approach. The developed algorithms are tested for limits and the tracking algorithm is compared to the previous implementations of DRAGON and *DimP*. Test are performed on artificial data, as well as several video scenes. Finally, chapter 7 — Summary and Future Work — contains a synopsis of our work and implications and ideas for future work dealing with direct manipulation.

# Chapter 2

# Related Work

*"Two weeks in the lab can spare you one afternoon in the library"*

*—Unknown*

Current implementations of direct manipulation were designed for different purposes and thus employ different kinds of visualizations and algorithms.

## 2.1 DRAGON

Karrer et al. [2008] focus on the current problems of video navigation and presents direct manipulation as a solution. Figure 2.1 shows the interface developed for direct manipulation. Clicking on a point will compute the trajectory for that point. Clicking on the blob allows interaction with the respective object. The interface presents several tools to deal with special situations. Trajectories show the path of objects, so even a user who does not know what exactly happens in the scene will know where to drag an object to achieve a certain goal. A clicking sound every ten frames represents the speed of the object in case the information is necessary. Finally, inertia allows to handle situations where the trajectory is ambiguous and errors of the algorithm. Trajectories become ambiguous when an object

Trajectory ambiguity


**Figure 2.1: DRAGON'S Interface:** The line shows the girl's path through the scene and the blob represents the current position of the interest point

passes the same point twice. Consider a swinging pendulum; the pendulum will pass the same point multiple times and thus this point is consistent with multiple points in time. Close to the turning points, the time difference is not a good heuristic to distinguish correct and incorrect points, because past and future points may be have equal time differences. Inertia allows the user to give the object a push in a certain direction, which is related to a direction in time, and it will continue to move into the same direction in time. The "harder" the user pushes the object, the longer it will move. This technique also allows to handle incorrect outputs of the tracking algorithm because an object will continue to move even after the computed trajectory ends. In

this way, the object can be pushed over an occlusion and grabbed again when the occlusion is resolved.

DRAGON uses an accurate optical flow algorithm by Weiss [2007] as basis for tracking. Optical flow is defined as "[...] the distribution of apparent velocities of movement of brightness patterns in an image" [Horn and Schunck, 1981]. A more common understanding of optical flow it the apparent movement of points in an image. This is different because optical flow computation assumes brightness constancy, while the latter only assumes some kind of matching between pixels. The optical flow algo-

**Figure 2.2: Tracking in DRAGON:** The white arrows are a (discrete) visualization of the flow field from this frame to the next

rithm employed implicitly deals with objects by imposing constraints on the movement of spatially related pixels. This allows for a very simple tracking algorithm at runtime that is only dependent on a single point. As soon as the user clicks on a point, the computed flow fields (see figure 2.2) dictate the movement of this point in consecutive frames. The approach allows for tracking of very small objects and is suited for very different kinds of videos. On the other hand, the algorithm is much too slow for real-time application of direct manipulation. Currently it takes approximately two minutes to process a pair of 640x480 video frames on one core of a Intel Xeon 2.8 GHz

DRAGON requires precomputed flow fields


**Figure 2.3:** ***DimP's*** **Interface:** The left image shows the interaction when the user stays close to the girl's path. The right image shows what happens when the cursor is farther away from the object.

quad-core processor. So even when employing all cores of a dual-processor machine, it takes about 15 seconds to compute flow in one direction. Hence, precomputing flow fields is necessary when using DRAGON, although it is infeasible in most situations.

## 2.2 *DimP*

Dragicevic et al. [2008] also directly deal with introducing direct manipulation to video scenes. The basic idea of their system *DimP* is the same as DRAGON. Only the visualization works a little different. Trajectories vanish when the mouse is close to the indicated trajectory (Figure 2.3). The user can click on any point in the video to start the interaction, but in regions that show relevant movement the mouse cursor changes to a hand to indicate interesting positions even in a still frame. *DimP* also considers moving cameras. This is necessary because "[...] people perceive


**Figure 2.4: Background stabilization in *DimP* [Dragicevic et al., 2008]:** A scene with an upward pan of the camera. The actual movement of the object in the scene (a), the corrected trajectory, which is now closer to the user's perceived motion (b), and the shift that occurs while dragging (c)

relative motion rather than absolute motion" [Dragicevic et al., 2008]. Relative motion is induced by surrounding objects in a scene. The player creates relative trajectories in respect to the camera motion and shifts frames inside its window in a way that the background of the original frame stays stable (Figure 2.4. Only translational movement of the camera is considered.

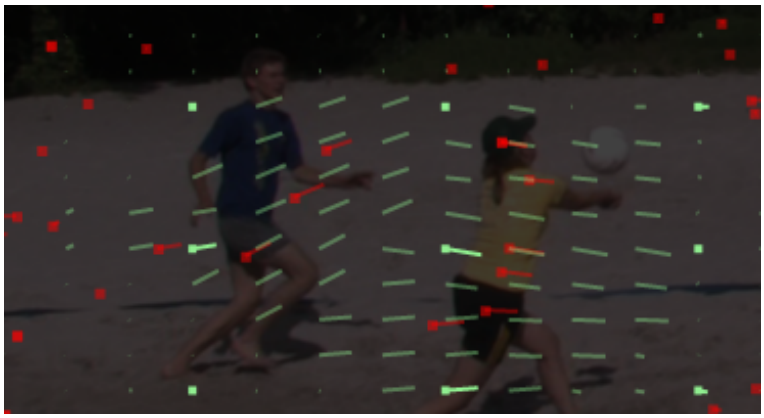

**Figure 2.5: Tracking in *DimP*:** The red dots represent SIFT features in this frame that could be matched to the next frame. The green lines visualizes the flow field interpolated from these features

*DimP* employs the same tracking technique as DRAGON. The system interpolates flow fields (Figure 2.5) based on

points detected and matched by the SIFT point detector (see section 3.3.2). The drawback of interpolating flow fields is loss in accuracy (see 6). Although this estimation approach is faster than DRAGON — tracking takes about 2.5 seconds for a pair of 128x128 frames on a single core — it is still not fast enough for real-time application.

## 2.3 *Trailblazing*

Video surveillance

Kimber et al. [2007] developed *Trailblazing*, a system for video-surveillance. One part of *Trailblazing* is a user interface, which allows tracking of persons over multiple cameras (Figure 2.6). Navigation in *Trailblazing* is done by direct manipulation. The user can click objects in the video stream and drag them along the trail shown by the system. It is also possible to manipulate a person's representation on a floor plan to interact with the video.

The current version of *Trailblazing* cannot support cam-


**Figure 2.6: Interface of *Trailblazing* [Kimber et al., 2007]:** Three views of the same hall. All can be used for navigation

era movement because the algorithm finds moving objects based on pixels changing between consecutive frames. Naturally, camera movement implies change in every pixel, preventing the algorithm from detecting foreground objects. *Trailblazing* is not able to distinguish objects occupying the same space, since a changing pixel is not related to a particular object. Especially a split of a moving region poses a problem to the system. To deal with this problem, all possible paths are shown to the user, leaving the decision for a direction up to him. Although this approach is sufficient for video surveillance with stationary cameras, it does not work for the general application of direct manipulation because of its limitations.

Unsuitable for camera movement

## 2.4 Schematic Storyboarding

The system described in Goldman et al. [2006] is designed to extract a storyboard from a given video scene. This storyboard contains information about movement of persons and cameras, which is represented by arrows placed inside a panoramic view of the scene (Figure 2.7). This visualization allows faster comprehension of video content than single frames. In addition to the storyboard visualization the information can be employed for direct manipulation. Clicking on the arrow moves the object to its respective position in the video and clicking and dragging the background allows to change camera position. Video and vi-

**Figure 2.7: Movement visualization by Goldman [Goldman et al., 2006]:** A scene is represented by important keyframes and 3D movement arrows representing object movement

sualization are separated in this approach. The developed system presented by Goldman does not implement an automatic tracking technique, but focusses on the interaction. Tracking has to be done manually for objects, as well as the background.

# Chapter 3

# Requirements for DragonEye

*"First, solve the problem. Then, write the code."*

*—John Johnson*

Before describing the developed tracking algorithm and the camera motion estimation, we define requirements for a tracking algorithm to be used for direct manipulation. These requirements are derived from the nature of the problem on the one hand and people who are supposed to work with the system on the other hand.

## 3.1 Technical requirements

Since DRAGON is supposed to work on arbitrary videos, the main prerequisite for tracking is the ability to work without previous knowledge of the objects contained in the scene. Thus, the object's size and shape, as well as all features required for tracking have to be determined dynamically. Objects in a scene usually have very different sizes. This is not as much related to physical size as to camera zoom and camera position. Size should not affect tracking accuracy and tracking speed should not change noticeably with entities of different sizes. Behavior , as well as appearance, are

Unknown object size and shape

Unknown object behavior


**Figure 3.1: Invariance to object size:** The bus and the couple in the background should be tracked equally fast

not known beforehand when using DRAGON. The tracker has to be able to handle objects moving toward the camera, effectively becoming larger, turning objects, and non-rigidity. The better the algorithm can handle these problems, the better it is suited for DRAGONEYE.

## 3.2 User requirements

User requirements are based on what people expect from a tracking system. These expectations are derived from how humans themselves follow objects. Although it is not realistic to expect the tracker to be as effective as the human visual system, limitations should be understandable by the user. One limitation will be produced by occlusions. Two basic kind of occlusions exist: self occlusion and scene occlusions. Usually full occlusions and partial occlusions are distinguished. During a full scene occlusion, the object is fully obstructed by another object in the scene. Partially occluding objects only hide a portion of the object of interest and self occlusions are occlusions by different parts of

Occlusion

the same object. The algorithm should at least be able to handle self occlusions and partial scene occlusions. Both are usually no problem for a human and should produce as few problems as possible. Because there is no way of telling when and where an object reappears after a full occlusion, the algorithm will not be able to handle full occlusions perfectly. In the case the entity shows unexpected behavior, like turning or changing its shape while hidden, the problem is not solvable without knowledge about possible states of the object. When continuing on the expected path, retaining shape and appearance, people will have no trouble finding the object again. The tracker should at least be able to track an object that did not change form, speed and direction while it was hidden. The better the prediction of behavior works, the better is the algorithm. An algorithm that considers recent behavior over a set of frames as well as the last known behavior will be better suited than an algorithm that only considers the most recent information. Informal user studies at out chair show that it is not necessary to follow the clicked point with absolute accuracy (see Figure 3.2), but only the containing object. People do not notice it when the tracked point moves from one end of the object to the other over several frames. This does not mean that the trajectory is allowed to show erratic behavior, but small inaccuracies are acceptable. Erratic behavior means that the interest point position moves back and forth on the object. This might hinder the interaction because trajectories become ambiguous, making frame accurate navigation harder. Precomputation of any sort is not feasible for direct manipulation. Video editing is often done under heavy time constraints. For example to create a replay of an interesting goal in the course of a soccer match. Since the main interest in this editing task is predominantly the position, the editor could benefit from direct manipulation. A precomputation time of more than a few seconds, will make direct manipulation infeasible in such situations. Reducing tracking times will cost accuracy of the algorithm. Up to a certain point this is acceptable, since very small objects, will mostly be of little concern to the user. The exact limits have to be determined through user tests.

Full occlusions are problematic

Absolute accuracy is not required

Erratic behavior hinders the interaction

We suspect that this is due to the fact that the point is not the locus of attention but the object itself is. This is supported by the result that not showing the tracked point after

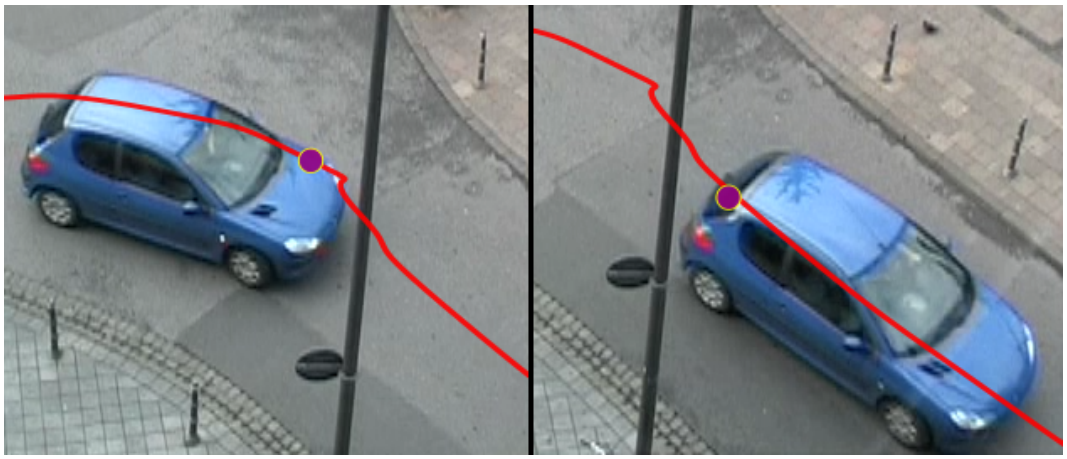

**Figure 3.2: Shift of the interest point:** The interest point shifted toward the back of the car during the course of the occlusion. Users do not usually notice this shift, because they are thinking about the car as a whole.

grabbing the object does not affect the interaction at all.

Definition:
*Locus of Attention*

**LOCUS OF ATTENTION:**
"Your locus of attention is a feature or an object in the physical world or an idea about which you are intently and actively thinking." [Raskin, 2000]

Since we do not know where the users locus of attention lies, the tracker needs to be versatile enough to accommodate for different loci of attention. For example, consider a scene, where a man walks down the street. In the initial frame, the arm is close to the torso. A user might be interested in tracking his arm, because he is going to pick something up, or the user might be interested in following the man down the street and does not care about the rest. It is not possible to guess the correct locus of attention, so it has to be indicated by the user.

Camera motion

Another important aspect that has to be handled by direct manipulation is background compensation. In lots of scenes, an object is followed by the camera. Most sports videos alternately show an overview and a detailed view of what is happening. In the detail view, the camera usually follows a specific player. This has the effect that the player stays static in the center of the video while the back-

ground moves. A person watching this video, would perceive the player as moving, since she understands the difference between player and background on a higher level. So, if she clicks on the player, she might expect a trajectory that shows the real movement of the player and not the static movement caused by the compensating camera movement. To handle this, DRAGON has to be able to represent the current camera movement in some way. A good example for this was implemented by Dragicevic et al. [2008], as discussed in section 2.2. Camera motion compensation should be an optional feature to still allow dragging the environment to "move" the camera.

## 3.3 Implications for the Implementation

DRAGONEYE has to fulfill at least the following requirements.

1. The tracker can handle objects of unknown color, shape and texture
2. The tracking algorithm can deal with rotation, size changes and form changes
3. Size does not noticeably affect tracking speed
4. Self occlusions do not affect tracking
5. The tracker can handle partial scene occlusions
6. Full scene occlusions can be handled in case the object shows predictable behavior
7. Tracking needs to be fast enough for real-time application
8. Support for the computation of camera motion

Arbitrary Objects

The fact that we have to be able to track arbitrary objects obviously makes it impossible to use any tracker requiring previous knowledge about tracked entities. Any model used has to be created from information given by the user.

In the optimal case this would only be the point the user clicked on, since this point is obviously part of the object that is supposed to be tracked. Unfortunately, a single point is not enough when considering the problems posed by occlusions. Occlusions usually pose a problem for all tracking algorithms. Handling full scene occlusions requires three phases. First, occlusion detection, second, behavior prediction, and third, recovery of the lost object. To do this effectively, a tracking algorithm for direct manipulation needs some kind of object model that can be used to reacquire the object after an occlusion. Partial scene occlusions can be handled if the part of the object that is still visible is reliable enough to continue tracking. In any other case we face the same problems as with full scene occlusions. Self occlusions can usually be handled better than scene occlusions. A tracker can replace the tracked part of the object with the occluding part. But this implies that the tracker knows that the occluding object is in fact another part of the occluded object. If this relationship is unknown, self occlusions pose similar problems as scene occlusions. Only a failure in occlusion detection would not be as dramatic (for the user), since the tracked object does not change.

Occlusions

Occlusions are a problem for simple flow-based trackers

The problem of occlusion is one of the reasons that simple objectless tracking like in the original DRAGON is not sufficient for direct manipulation. Occlusions cannot be detected deterministically if the tracker has no notion of objects. A more detailed discussion of the limits of the current approaches can be found in chapter 5.

Object Size

Point three implies that the algorithm is either so fast that changes in size do not noticeably affect speed, or the the accuracy of the algorithm is not significantly reduced if image size is adapted according to the current object size.

A lot of research has been done in the field of object tracking. In the following, we try to give a short overview of important object representation and tracking techniques and evaluate their usefulness for direct manipulation. For a detailed review of object tracking, we refer to Yilmaz et al. [2006] (broad review of object representation techniques and tracking algorithms) and Mikolajczyk and Schmid [2005] (comparison of point tracking algorithms).

### 3.3.1 Kernel Tracking

A template is a description of an image region, also called kernel. Content of this kernel is described by color distribution [Fieguth and Terzopoulos, 1997], texture [Comaniciu et al., 2003], or other descriptive features. The idea of a kernel tracker is to find the region in an image that matches this template best. Kernel tracking has the advantage that objects can be represented by different kinds of features fitting the occasion.
Fieguth and Terzopoulos [1997] use color to describe an image region. A region is rectangular and receives a descriptor based on the average color in the RGB color space

$$(r, g, b) = \frac{\sum_{i=0}^{n}(r_i, g_i, b_i)}{n}$$

for every pixel in the region. The region is tracked by finding a new center position for a new region of the same size, that best matches the old descriptor based on finding the optimal region of the same size in the next image. The optimal region is the region centered in $(x, y)$ where $\psi(x, y)$ is minimal.

$$\psi(x, y) = \frac{\max(\frac{r}{r'}, \frac{g}{g'}, \frac{b}{b'})}{\min(\frac{r}{r'}, \frac{g}{g'}, \frac{b}{b'})}$$

Brute force search

This approach is very simple, but requires a brute force search for the optimal position. It allows for fast tracking if only a limited number of positions is tested. The latter can be achieved by assuming only small changes from one frame to the next and prediction techniques like a Kalman filter (see section 3.3.4). The authors also propose an explicit occlusion model to deal with obstructed objects. The approach only considers translational movement and can only deal with limited change in size or rotation.

CAMShift

Continuously adaptive mean shift (CAMShift) presented by Comaniciu et al. [2003] does not require a specific object representation. Any representation that can be expressed by a probability density function (pdf) will work. The implementation by Comaniciu and the implementation contained in OpenCV [Ope] use a color histogram as a discrete estimate for the pdf. Comaniciu employs histograms in RGB color space while the OpenCV sample implementation uses the hue component of the HSV color

CAMShift can work with any probability function

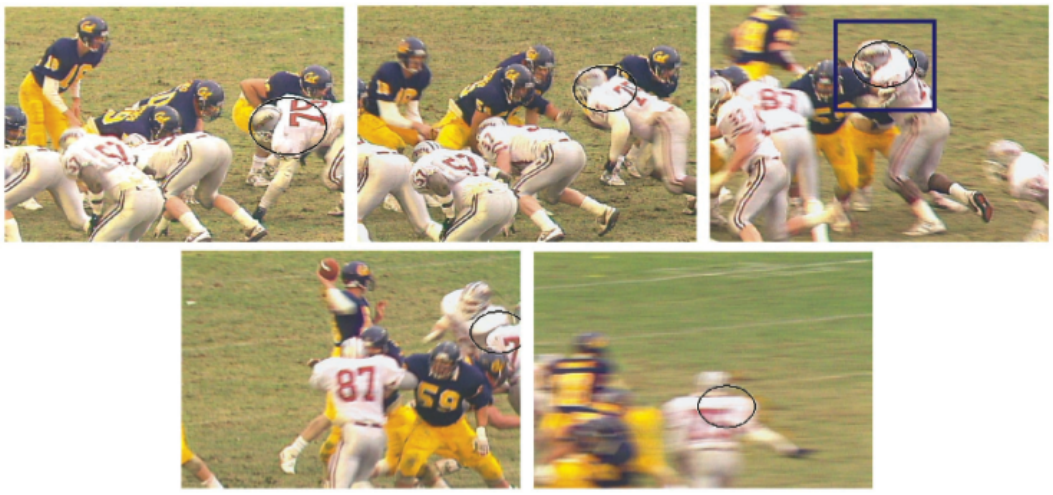

**Figure 3.3: CAMShift [Comaniciu et al., 2003]** An image sequence containing occlusions and heavy motion blur because of fast camera movement

Iterative approach

space. Instead of using a brute force approach to find the best matching center position, the position is changed iteratively until converging on a point. The change in each iteration is determined by the mean-shift approach Comaniciu and Meer [2002]. When using mean-shift, the next position of a kernel is determined by the so-called mean-shift vector. The authors propose to use special kernels with Epanechnikov kernel profile Comaniciu and Meer [2002]. A radially symmetric kernel has a Epanechnikov profile if the weight of a pixel in the kernel can be expressed by the distance to the center.

$$k(x) = \begin{cases} 1 - x & 0 \leq x \leq 1 \\ 0 & x > 1 \end{cases}$$

When using such a kernel, the mean-shift vector $\vec{m}$ is defined as

$$\vec{m} = \frac{\sum_{i=1}^{n_h} x_i w_i}{\sum_{i=1}^{n_h} w_i} - y$$

where $y$ is the kernel center, $x_i$ are the pixel locations in the kernel, and $w_i$ are weights for each pixel based on the quality of the template match.

$$w_i = \sum_{u=1}^{m} \sqrt{\frac{\hat{q}_u}{\hat{p}_u(\hat{y})}} \cdot \delta[b(x_i) - u]$$

$\hat{q}_u$ defines the probability of a feature $u$ in the original model, $\hat{p}_u$ is the probability in the tested region, $\delta$ is the Kronecker delta function and $b(x)$ defines the mapping of a pixel $x$ to a feature class.
Intuitively, a pixel $x_i$ is weighted higher if it was mapped to a feature class that fits the model. Thus, the center is shifted towards such better fitting pixels. CAMShift can deal with size changes by testing multiple kernel sizes. The authors propose three tests with 10% larger or smaller target kernels. A good value for the maximal number of shifts is 20. Usually not more than four are required before convergence is reached. CAMShift based on color histograms produces good results in the presence of motion blur (Figure 3.3), which is often a problem for different algorithms because of their reliance on stable gradients.

The resistance to motion blur is a good argument for using kernel tracking with color histograms in DRAGONEYE. Since arbitrary videos have to be considered a tracker cannot rely on a single feature type for object tracking. Also, direct manipulation requires a single point that can be used as a reference; even if the visualization worked on objects and not points because the interaction needs to be. The representation as a kernel might proof to be too unstable for effective point tracking. Constant changes in size and form will introduce erratic changes in the trajectory which might be acceptable during a motion blur but might hinder the interaction when positions on the trajectory become ambiguous because of this.

### 3.3.2 Point Tracking

Objects can also be represented as a set of points, which are subsequently used to determine an object's position, and sometimes pose. These sets can be simple and only contain a single point, for example a centroid, or they can contain many points, chosen to represent different parts of the object [Serby et al., 2004]. The tracking mechanism in DRAGON is an example for the first approach. An is represented only by a single point; namely the point the user clicked on.
Tracking points requires two algorithms; one to find points

Harris Corner Detector

and one to match points onto each other. The Harris corner detector Harris and Stephens [1988] focusses on finding corner like structures. Corners have the property that they show both significant horizontal as well as vertical image gradients. These gradients are represented in the eigenvalues of the local autocorrelation matrix, since it contains information about changes in the neighborhood of the point. If both eigenvalues are small, the image does not change much in the local neighborhood. A large difference in the eigenvalues means, that the autocorrelation function shows large changes in one direction, which indicates an edge. If both eigenvalues are high, the autocorrelation function shows changes in both directions, indicating a corner. The local autocorrelation matrix $C$ can be estimated using the first order tailor expansion, yielding

$$C = \sum_{(x,y)\in W} \begin{pmatrix} I_x(x,y)^2 & I_x(x,y)\cdot I_y(x,y) \\ I_x(x,y)\cdot I_y(x,y) & I_y(x,y)^2 \end{pmatrix}$$

where $I_x$ and $I_y$ are the partial image derivatives and $W$ is the autocorrelation window.
The original Harris detector is only invariant to rotation. It has since been extended by Mikolajczyk and Schmid [2001] to handle scale changes as well. Robustness against scaling is achieved by detecting points in several versions of the image, each representing different object sizes.

Scale space describes signals at different scales

Handling size changes in an image is often done using a multi scale representation of the image Lowe [2004], Mikolajczyk and Schmid [2001]. This multi scale representation, also called image pyramid, is described by the scale space theory by Witkin [1983], Lindeberg [1991]. Scale space is defined on signals as "[...] *embedding* of the original signal into a one-parameter family of derived signals constructed by convolution with a one-parameter family of Gaussian kernels of increasing width." Lindeberg [1994]. Images can be represented as two-dimensional discrete signals, so scale space theory applies to images as well. Figure 3.4 shows an example of a smoothed signal. An example of Gaussian smoothing in an image can be found in figure 4.7 in chapter 4. Both examples show how details are discarded with increasing width of the Gaussian kernel. Lindeberg showed that only the Gaussian kernel and its derivatives, especially the Laplacian of the Gaussian $\Delta f$ — the sec-

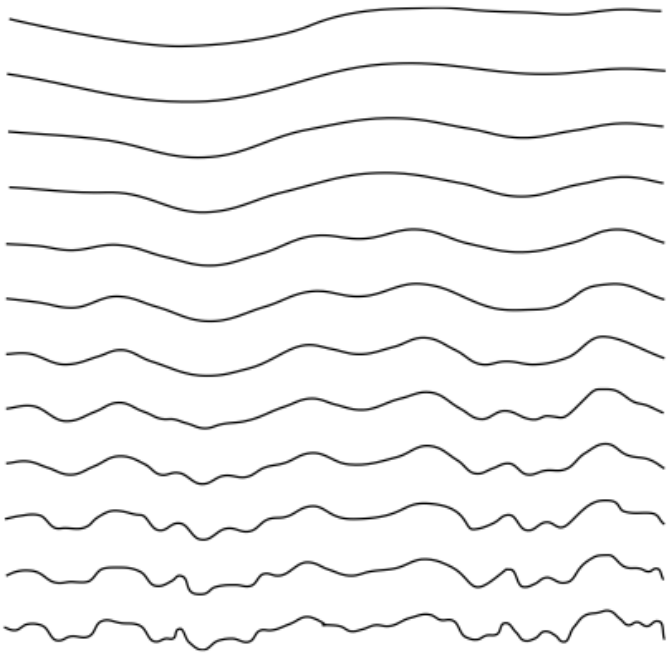

**Figure 3.4: Scale space representation [Witkin, 1983]:** A one dimensional signal convoluted with several Gaussians (Eq. 3.1) of increasing width.

ond order derivative — can be employed to analyze images in scale space. Zero crossings and extrema of the signal smoothed with the Laplacian determine the position of corners and blobs, respectively. Even without the notion of stable points in scale space, the pyramid approach has been applied for several decades to analyze images.

Stable points in an image can be found using the Laplacian

$$G(x,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{3.1}$$

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.2}$$

$$G(x,y,\sigma) = G(x,\sigma) \cdot G(y,\sigma) \tag{3.3}$$

$$\Delta f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} \tag{3.4}$$

Lowe [1999, 2004] introduces SIFT which is based on Lin- SIFT

debergs finding of stable points in scale space. SIFT is designed to be robust to rotation, scale changes, and illumination changes. A keypoint is detected by SIFT, when it is an extremum in a image pyramid built of difference images between gaussian convoluted images. Extrema in this pyramids are extrema in a 3x3x3 environment (Figure 3.5). This directly relates to extrema of the Laplacian because it is approximated by the DoG (Difference of Gaussians). In

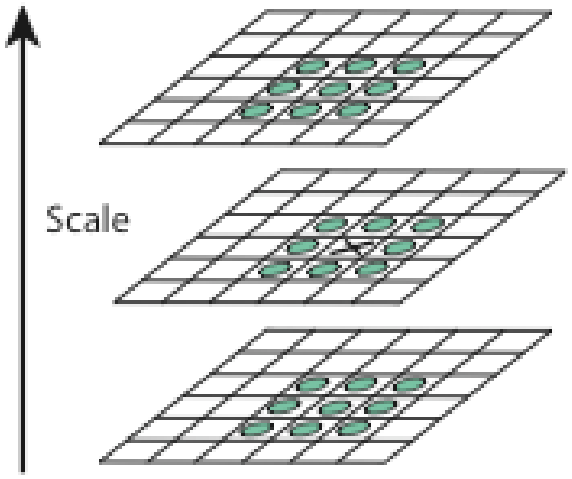


**Figure 3.5: SIFT keypoint detection [Lowe, 2004]:** If the sample in the center level is a maximum of all eight surrounding points, it is a keypoint candidate.

contrast to the Laplacian $\Delta f(x, y)$ (Equation 3.4), the Gaussian $G(x, y, \sigma)$ is separable (Equations 3.1 – 3.3), allowing for faster computation. Since the Gaussian images are required for descriptor computation later on, the difference of Gaussians is easily obtained by computing the difference image of two adjacent convoluted images. The $\sigma$ for the Gaussians are chosen to produce a multiple of two after $s$ convolutions. These $s$ convolutions are called an octave. In this way, the image can simply be resampled at the end of an octave using every second pixel. This drastically reduces computation time at higher levels, since image size is halved every octave.

Not all points found in this way are suitable for stable representation of an object. Low contrast points and points on edges are usually susceptible to even small amounts of noise. Points lying on edges, i.e. showing large differences of the eigenvalues of the Hessian matrix, and low contrast responses are eliminated. The Hessian matrix contains second order derivatives, but the edge suppression principle is similar to the autocorrelation matrix used for the Harris detector.

Point description and matching techniques

Matching of points in two different images requires descriptors. Existing descriptors are based on several characteristics, including pixel intensity, gradients or position of surrounding points. A very simple descriptor consists of only pixel intensities (or colors) which can then be matched using cross correlation. The dimension of such descriptor vectors grows very fast with the size of the considered patch around the detected point and thus require a high computational effort to match. Intensities are also very susceptible to illumination changes. Normalized gradients, as applied by Lowe [2004], are invariant to affine — of the form $a \cdot I + b$ — changes in illumination. Lazebnik et al. [2003] propose a distance based descriptor which classifies pixels surrounding the interest point by distance and intensity. In a comparative study, Mikolajczyk and Schmid [2005] showed that the SIFT descriptor is only outperformed slightly by GLOH (Gradient Location and Orientation Histogram [Mikolajczyk and Schmid, 2005], which is an extension of the SIFT descriptor. A good descriptor will be very helpful in case of an occlusion. If points can be distinguished very accurately, occlusions can easily be detected because a portion of the tracked points is lost. Also reacquiring the object after an occlusion is easier if points are unambiguous.

SIFT descriptor

The SIFT descriptor is always computed on the level of the Gaussian pyramid closest to the pyramid level of the DoG pyramid on which the corresponding point was found. A Gaussian circular window defines the pixels used to describe the point. The size of this window increases with the height of the pyramid. This is what makes the the descriptor resistant to scaling when the area around the point scales conforming to the point itself. To make the descriptor invariant to rotation, the main orientation of the point is

computed. Orientations of the surrounding points, which are later on used as the descriptor, are then only stored relative to the main orientation. Equation 3.5 is used to compute the local orientation of a point in an image $L$.

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \tag{3.5}$$

$$m(x,y) = \big[(L(x,y+1) - L(x,y-1))^2 + (L(x+1,y) - L(x-1,y))^2\big]^{\frac{1}{2}} \tag{3.6}$$

The second equation 3.6 describes the magnitude of the local image gradients. Orientation of samples in a region around the point are weighted by the magnitude and a Gaussian with $\sigma = 1.5$ time the level-$\sigma$. The results are used to form a histogram with 36 bins — a bin represents 10 degrees. The mode of this histogram then defines the orientation of the keypoint. Should a second bin of this histogram come within 80% of the mode, multiple orientations are assigned to increase stability.

The descriptor is formed in a similar way. 16 Histograms with only eight instead of 36 bins are formed in a 16x16 sample window. Each histogram corresponds to a 4x4 sub window. Separating the histograms in smaller blocks allows for non-rigidity and affine transformations such as shearing. Gradients are allowed to shift up to four samples while still belonging to the same histogram. Also, changes in one histogram do not necessarily effect the others. All these histograms define the descriptor. It is normalized to be invariant to affine changes in illumination. Truncation to 0.2 and a second normalization produce resistance to non-linear changes in illumination. Such changes more commonly affect gradient magnitudes and leave the orientation untouched. The truncation reduces the influence of large magnitudes and increases the importance of orientation.

Descriptor matching

The difference between two SIFT descriptors is defined by the euclidian distance in 128 dimensional space. Reliable matching of two descriptors requires a way to eliminate false matches. Lowe introduced an approach based on tests with the ratio of the distance to the best match and the

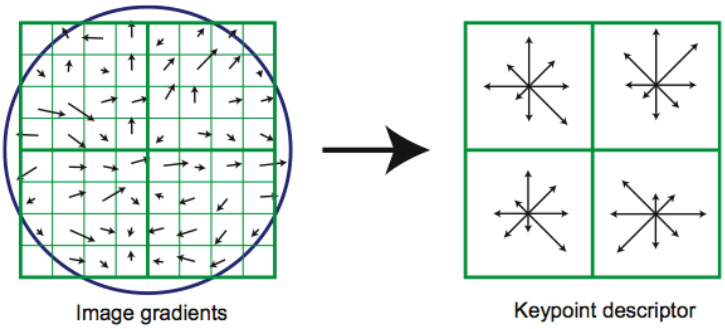


**Figure 3.6: SIFT descriptor computation [Lowe, 2004]:** A simplified example of descriptor computation with 4 histograms in a 8x8 by sample window. The circle indicates the gaussian. The orientations, weighted by magnitude, indicated by length of the arrows are assigned to 8 histogram bins on the right, yielding 4 histograms.

distance to the second best match. Matches with a ratio smaller than 0.8 are discarded. Thus, 90% of false matches and only 5% of correct matches are rejected 3.7.

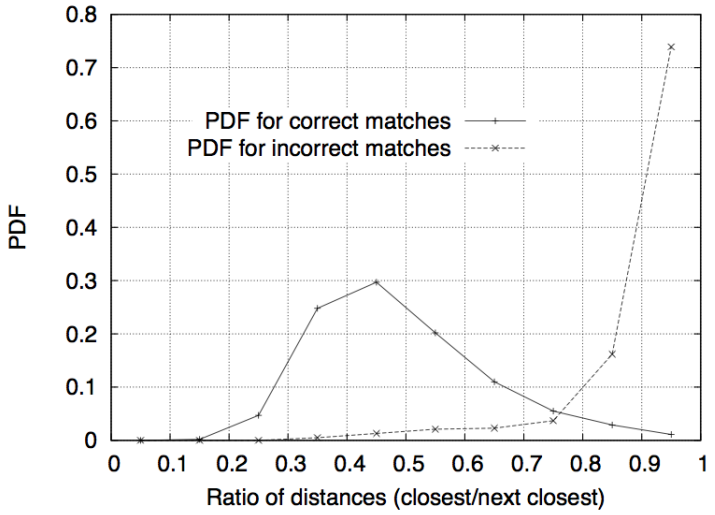


**Figure 3.7: SIFT matching [Lowe, 2004]:** The probability distribution of the ratios between distance to best and second best match in a database of 40000 descriptors

Besides simple point tracking with SIFT as in *DimP*, Tang SIFT tracking

and Tao [2005] propose a tracker defining objects as graphs. Position and pose of the objects is then determined by graph matching. The composition of the model graph is defined by an m-order Hidden Markov Model. Features are added and removed from the model based on the binomial distribution. A feature is added when it was observed $n$ times over the last $m$ frames and $B(n; m, p) > \tau$, where B is the binomial distribution, $p$ is the probability for observing the feature and $\tau$ is the threshold for adding. Features are thrown out if the binomial distribution falls below a second threshold. To simplify the NP-complete problem of graph matching, only a subgraph of the complete image graph is searched for the model.

### 3.3.3 Silhouette Tracking

Complex non-rigid objects are usually hard to track by means of point or kernel tracking. Kernel tracking requires a predefined kernel in which to look for the object and points descriptors are based on surrounding characteristics as well. An object does not necessarily follow these concepts. For example, an object might have a form that does not fit the kernel and thus the kernel is is either dominated by the background or may become too small for stable tracking. Point descriptors might be thrown off by changes in contour affecting gradients or the arrangement of surrounding points. Silhouette tracking defines an object by this surrounding shape and can handle changes better. Yilmaz et al. [2006] classifies silhouette trackers into two different categories. Shape matching and contour tracking. The first approach tries to fit a shape model from previous frames to a new frame, while the second approach evolves the contour from one frame to the next. Shape matching does not explicitly handle shape changes. Instead parts of the object that are not affected by shape changes are emphasized. For example, a walking person, would normally show large changes in arm and leg shape, while the torso and head do stay stable. Huttenlocher et al. [1993] assumes that objects may exhibit only small changes in shape, while position may change drastically. Shapes are defined using edge maps [Canny, 1986] and represented using binary images. Matching of shapes is done by computing an adapta-

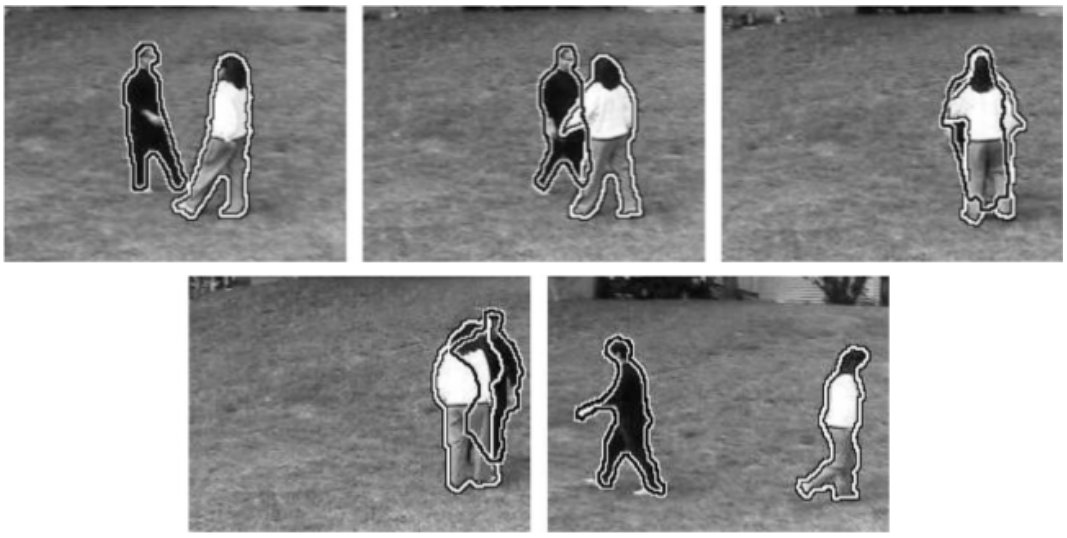

**Figure 3.8: Silhouette tracking [Yilmaz et al., 2004]:** An example of a silhouette tracking with sophisticated occlusion handling

tion of the Hausdorff distance between two sets of points. The Hausdorff distance is defined as.

$$H(P,Q) = \max(h(P,Q), h(Q,P))$$

$$h(P,Q) = \max_{\mathrm{p}\in\mathrm{P}} \min_{\mathrm{q}\in\mathrm{Q}} ||p-q||$$

The distance $d$ between two shapes is then defined a

$$d(P,Q) = \min_{\mathrm{g}\in\mathrm{G}} H(g(P),Q)$$

for a set of allowed transformations $G$. Allowed transformations, such as translation or rotation, can be defined according to the application. Intuitively this means that a shape matches another shape well when each point (on the edge map) is close to a point on the other edge map. The Hausdorff distance has been adapted to reduce the strong influence of outliers. Li et al. [2001] employs shape matching to estimate the pose of objects in different frames. The authors propose to use another version of the Hausdorff distance for matching. Objects are also represented using edge maps. Additionally, they proposes to compensate background movement which is applied to reduce the occurrence of background points in the model. Contour tracking explicitly deals with changes in shape by evolving a contour from one frame to the next. In contrast to

shape matching, contour evolution does not require the object to have stable parts, but it is important that the position does not change too much between frames to have a valid starting point for the evolution. Yilmaz et al. [2004] developed a contour tracker based on an energy functionals, that maximizes the probability of a pixel belonging to the background or the object. The probabilities are defined by color and texture features learned in previous frames. Moreover, Yilmaz introduces a probabilistic shape model that allows to estimate the object contour in the case of an occlusion. Silhouette tracking performs very well in case of rigid and non-rigid objects. On the other hand, it relies on a good definition of an initial contour that defines the first model. Models created by background subtraction cannot handle different objects in the same area, and will fail in the presence of camera movement. When using background stabilization combined with background subtraction, the initial model relies very heavily on correct camera motion estimation. Hence, silhouette tracking cannot be applied to direct manipulation directly because shape definition would be a large burden for the user. Also, the knowledge about the complete contour might actually be too much, and unnecessary for the interaction.

### 3.3.4 Occlusion Handling

The presence of occlusion in video scenes will be a problem that cannot be avoided in direct manipulation. It has to be dealt with explicitly. Especially full scene occlusions pose a problem since they have to rely on prediction techniques. These prediction can be based on the relation between different objects in the scene or on recent behavior of the object. A common prediction techniques is the Kalman filter [Welch and Bishop, 2001, pp. 20 - 22], which tries to predict the state of an object based on the previous behavior. This behavior is encoded in a matrix A, which is updated in every step, effectively doing an iterative least squares. The error that is minimized in this case, is the difference of the predicted and correct state.

# Chapter 4

# Camera Motion Estimation

As mentioned in section 3.2 knowledge of the effects of camera motion is necessary to support the users greater understanding of a scene. With this information, we are able to show visualizations such as panoramic views of the whole scene or cutouts of the panoramic as in *DimP*. This is also called background stabilization, because the previously moving background appears still. Besides showing the stabilized scene, the system also needs to present a trajectory which is altered according to camera motion. Camera motion can be modeled in several ways. The simplest idea is to only consider translational movement. Only a single vector would describe the movement for all pixels in the video. This vector could be extracted by computing the average movement over all known points, or with more sophisticated approaches considering different movement induced by background and foreground objects. *DimP* [Dragicevic et al., 2008] is an example of a system that uses a translational movement model for background stabilization. The representative vector is found by greedy binary segmentation of the space of movement vectors, yielding the most dense region in the space. Unfortunately , camera motion is much more versatile than translation. For pans, which technically are not a translation, the error introduced by only considering translation between individual frames is small (Figure 4.1). Errors start to be visible with

Translational model

increasing distance in frames (Figure 4.2). Zooms and rotations, on the other hand, cannot be reproduced by a single motion vector (Figure 4.3). Since especially zooms are quite usual in video scenes, direct manipulation requires a more complicated motion model, which can handle these effects.


**Figure 4.1: Translational camera model (Pan):** 20 frames of a pan related by a translational movement model. Created with Photoshop CS3

**Figure 4.2: Translational camera model without Intermediate Frames:** The error of the translational model becomes clearer when intermediate frames are left out. Created with Photoshop CS3

Affine model

Affine transformations represent more complex movement, including translation, zoom and rotations. An affine map is defined by two components; a linear map $A$, and a translational component $\vec{v}$ (Equation 4.1). In computer vision, lin-

**Figure 4.3: Translational camera model (Zoom + pan):** 20 frames of a pan and zoom related by a translational movement model. The zoom part of the camera movement cannot be visualized. Created with Photoshop CS3

ear maps define combinations of scale, rotation and shear, while translation can obviously not be expressed linearly.

$$p' = A \cdot p + \vec{v} \tag{4.1}$$

A two dimensional affine map contains six parameters

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a_{13} \\ a_{23} \end{pmatrix}$$

or in homogeneous coordinates

$$\begin{pmatrix} x' \\ y' \\ w \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

Thus, an affine map is a special case of a homography without perspective component (see equation 4.3). Estimating an affine model requires at least three points to work. A point has two components, so with three points we have the six values necessary to solve the linear equation system with six unknowns. More points are better since it allows for compensation of errors by point detection. The overdetermined system arising from using more point can subsequently be solved using least squares (see 4.1), assuming that this error is distributed normally.

Estimating affine models takes 3 points or more

Affine models cannot represent changes in perspective. These are taken into account by a homography. Homogra-

phies, also called perspective mappings, are usually used in mosaicking systems, such as

Projective mappings from a point $\begin{pmatrix} x & y \end{pmatrix}^T$ to a point $\begin{pmatrix} x' & y' \end{pmatrix}^T$ are defined as

$$x' = \frac{h_{11} \cdot x + h_{12} \cdot y + h_{13}}{h_{31} \cdot x + h_{32} \cdot y + h_{33}} \tag{4.2}$$

$$y' = \frac{h_{21} \cdot x + h_{22} \cdot y + h_{23}}{h_{31} \cdot x + h_{32} \cdot y + h_{33}}$$

where $h_{11}$ to $h_{33}$ are the parameters of the mapping.

For two dimensions it can be written as a 3x3 matrix which looks as follows.

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \tag{4.3}$$

When applying a homography to a point $P_C = \begin{pmatrix} x & y \end{pmatrix}^T$ in cartesian coordinates, we transform it to homogeneous coordinates $P_H = \begin{pmatrix} x & y & 1 \end{pmatrix}^T$ and multiply $H$ with $P_H$

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = P'_H = H \cdot P_H$$

To convert a point in homogeneous coordinates back to cartesian coordinates, we divide all components by the third, perspective, component and use the first two components as position components.

$$P'_C = \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \end{pmatrix}$$

Combining two homographies is as simple as multiplying the two respective matrices, yielding another homography. For camera movement, this means, that the camera movement from frame $x$ to frame $x + t$ can be computed by multiplying the $t$ homographies for the intermediate frames.

$$P' = (H_x \cdot H_{x+1} \cdot ... \cdot H_{x+t-1}) \cdot P$$

$$P' = H_{x \to x+t} \cdot P$$

The same matrix $H_{x \to x+t}$ can be applied to all points.

Unger [2004] uses a biquadratic model containing twelve parameters to distinguish fore and background in a video scene.

$$x' = b_1 \cdot x + b_2 \cdot y + b_3 \cdot x^2 + b_4 \cdot y^2 + b_5 \cdot x \cdot y + b_6$$

$$y' = b_7 \cdot x + b_8 \cdot y + b_9 \cdot x^2 + b_{10} \cdot y^2 + b_{11} \cdot x \cdot y + b_{12}$$

The qudratic dependency of the coordinate prevents the combination of two models to a model of the same complexity, which requires the combination of models for each point individually.

$$P' = B_{x+t-1}(...B_{x+1}(B_x(P))...)$$

This increases the complexity of computing the combined motion over several frames.

All the mentioned models consider the background to be planar, which is obviously not the case in most video scenes. The assumption fails as soon as the center of projection starts to differ. Although the assumption is incorrect, the resulting mosaics look good, which is sufficient for visualization [Dragicevic et al., 2008, Hsu et al., 2000, Irani et al., 1995]. The reason for this is that the background structure forms a (virtual) plane. In many cases, this virtual plane represents the background sufficiently.Problems arise when several background structures form completely different planes. Most algorithms will choose the larger plane and thus the other plane will show ghosting artifacts (Figure 4.4). Determining the exact camera motion, or its effects, requires us to know more about the underlying 3D structure of a scene. Although this structure can be computed from three images of the same frame [Hartley and Zisserman, 2000, pp. 262 – 278], the process is computationally more expensive. Inaccuracies are a problem too, since estimating the 3D geometry requires us to guess the intrinsic camera parameters as well. Chon et al. [2007] describes a specialized solution for mosaicking roadside buildings filmed from a moving car. In spite of the problems of multiple planes, we believe that the usually the largest plane is of the most interest to the user. Thus, considering the time constraints for direct manipulation, a planar homography should be enough for direct manipulation.

**Figure 4.4: Ghosting artifacts [Chon et al., 2007]:** An example of mosaicking with extreme ghosting artifacts

Finding such a homography is a well researched topic in the field of computer vision and is done with the DLT algorithm [Hartley and Zisserman, 2000, pp. 91, 109, 592]. It requires a set of sample points, that can be used to solve a linear equation system. We try to find an homography that matches points $P_i = \begin{pmatrix} x & y & 1 \end{pmatrix}^T$ in the first image to points $P_i' = \begin{pmatrix} x' & y' & 1 \end{pmatrix}^T$ in the second image.

Since a scalar factor for this matrix does not effect the resulting coordinates (see 4.2), we can scale the matrix by a factor $\frac{1}{h_{33}}$. This would only pose a problem if $h_{33}$ was zero which can only happen in the special case that $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ maps to infinity. Therefore, we can assume $h_{33}$ to be one. This yields the matrix

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$$

which has eight unknowns. Thus, we require four points of which we know the position in both images to compute the homography.

## 4.1 Least Squares

Although four points are theoretically enough to compute a homography, we have to consider the problems arising from errors caused by the point detection step. Hence, we have to consider more than four points. To solve the

overdetermined linear equation system given by more than four points, DLT uses least squares. For each point $P_i = (x_i, y_i, w_i)^T$ and a correspondence $P_i' = (x_i', y_i', w_i')^T$ we get

$$P_i' \times HP_i = \begin{pmatrix} y_i' \mathbf{h}^{3T} P_i - w_i' \mathbf{h}^{2T} P_i \\ w_i' \mathbf{h}^{1T} P_i - x_i' \mathbf{h}^{3T} P_i \\ x_i' \mathbf{h}^{2T} P_i - y_i' \mathbf{h}^{1T} P_i \end{pmatrix}$$

where $h^{iT}$ denotes the i-th row of the result homography H. Consequently we get the 3x9 matrix $A_i$

$$A_i = \begin{pmatrix} 0 & x_i w_i' & -x_i y_i' \\ 0 & y_i w_i' & -y_i y_i' \\ 0 & w_i w_i' & -w_i y_i' \\ -x_i w_i' & 0 & x_i x_i' \\ -y_i w_i' & 0 & y_i x_i' \\ -w_i w_i' & 0 & w_i x_i' \\ x_i y_i' & -x_i x_i' & 0 \\ y_i y_i' & -y_i x_i' & 0 \\ w_i y_i' & -w_i x_i' & 0 \end{pmatrix}^T$$

with

$$A_i \cdot \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = 0$$

When solving for H the 3rd equation in $A_i$ is usually omitted, yielding a new 2x9 matrix $A_i$

The $A_i$ are subsequently assembled into a 2$n$x9 matrix A.

$$A = \begin{pmatrix} A_0 \\ A_1 \\ \cdot \\ \cdot \\ A_n \end{pmatrix}$$

The (overdetermined) system

$$A \cdot \mathbf{h} = 0$$

with the constraint $||\mathbf{h}|| = 1$ to avoid the trivial solution, can then be solved by employing least squares. An exact

solution generally does not exist. Instead a norm $||A\mathbf{h}||$ is minimized, which is done using the singular value decomposition of $A$.

Singular value decomposition

The SVD (Singular Value Decomposition) of a matrix A is defined as

$$A = UDV^T$$

with $U$ and $V$ orthogonal matrices and $D$ a diagonal matrix with non-negative entries. The entries in $D$ are called singular values of a matrix A and are the square roots of the eigenvalues of the matrix $A^TA$

After applying the SVD to $A$, we need to minimize $||UDV^T\mathbf{h}||$. Since U and V are orthogonal, it holds that $||UDV^T\mathbf{h}|| = ||DV^T\mathbf{h}||$ and $||\mathbf{h}|| = ||V^T\mathbf{h}||$. Substituting $V^T\mathbf{h}$ with $\mathbf{y}$ yields the problem of minimizing $||D\mathbf{y}||$ with the condition $||\mathbf{y}|| = 1$. Thus, minimizing $||D\mathbf{y}||$ requires $\mathbf{y}$ to have a 1 at the position of the smallest singular value in $D$, which yields $\mathbf{h} = V\mathbf{y}$. This is also the eigenvector corresponding to the smallest eigenvalue of $A^TA$.

## 4.2 RANSAC

Least squares is very susceptible to outliers. These are caused by errors in the matching algorithm or simply by points on an object not belonging to the background. Also, we assume points to be located on a single plane, which is not the case for most images, making points on one plane outliers for the other planes. Thus, we are required to use an outlier suppression technique. RANSAC, short for random sample consensus, is such a technique. The principle of RANSAC is simple

1. Take a random sample of sufficient size from all data points
2. Use the sample to compute the model
3. Terminate if the enough of the other data points fit the model or go to step 1

In the case of estimating a homography we use four points to compute $H$. To determine how well it fits all the other points, we use the algebraic error

$$e_a = \sum_{i=0}^{n} ||H \cdot P_i - P_i'||_2$$

As discussed in Chum and Pajdla [2002], this is not the best error function since it does not account for errors in the first frame and is dependent on the homography. On the other hand, better error functions like the geometric error, which consider errors in both frames are more costly in terms of computation time. It has been shown that the algebraic error works well enough when appropriate normalization is employed (see [Hartley and Zisserman, 2000, p. 109]).

## 4.3 Implementation

we used the implementation of the DLT algorithm provided by OpenCV [Ope]. RANSAC was implemented in Objective C and Cocoa. Visualization of the estimated homographies was done with Core Image (see Appendix A.1).

we first tried to use motion vectors used in standard video compression algorithms. These motion vectors typically define the rough movement of so-called macro blocks between images. In this way, only differences between the corresponding macro block pixels in the old and new frame have to be encoded. The vectors for the macro blocks usually fit well, when they correspond to the underlying object and camera motion (Figure 4.5). Unfortunately, the algorithms are not designed to find the correct motion, but rather vectors that fit the compression technique. Thus, the vectors corresponding to the camera motion are not always found. Especially with a fast moving camera, the vectors often do not fit the camera motion as seen in figure 4.6. Finally, we decided to use points detected by SIFT. We used the GPU implementation by Wu.

we chose to allow a maximum of 150 iterations of the RANSAC algorithm. The iteration stops early if more than

**Figure 4.5: Suitable MPEG motion vectors:** Motion vectors fitting the camera motion. Extracted from MPEG4 encoded video

**Figure 4.6: Unsuitable MPEG motion vectors:** Motion vectors not fitting the camera motion. Extracted from MPEG4 encoded video

70% of the features fit the model. If less than 40% of features can be used for estimation, the algorithm returns no homography. In this case, the homographies in a radius of three frames are considered, assuming camera motion does not change much between frames. In a typical movie scene foreground objects are in camera focus and thus, show more detail than background objects. Low level features are directly related to detail, because they are found in very small regions. Since these detail features more often lie on

foreground objects 4.7, they are of no concern for camera motion estimation. In fact, if too many of such foreground detail features exist, the camera motion estimation will be influenced, resulting in more RANSAC iterations or incorrect estimations of the background. To avoids this effect, features for camera motion estimation should originate on higher pyramid levels, which usually provide a more suitable object to background ratio. In videos taken with camcorders designed for home use, the depth of field is usually larger. In this case, foreground objects do not necessarily show more features than the background. Since using higher pyramid levels reduces the number of features in both parts, this is also a wanted effect. Less features simply mean faster download of features from graphics memory and faster testing of compliance to other features. In case the background was very lightly textured and does not show many features, this also makes no difference, because well textured foreground objects will dominate the computation on any level.

Unfortunately the SIFT implementation does not allow to change the starting octave dynamically, which is necessary for object tracking (see 5.4). Thus, images are scaled down to a size of 64x64 before feature extraction, which has the same effect. Downscaling uses CGImage in the Apple Quartz API with high interpolation enabled. To allow for faster creation of trajectories, the algorithm starts computing camera motion as soon as the user stops the player. This is possible, since the camera motion is not dependent on a trajectory. Then, as soon as the user clicks on a point, only the object has to be tracked, which allows for a fast start when computing trajectories. More details on the combined implementation of camera motion and object tracking can be found in section 5.4.

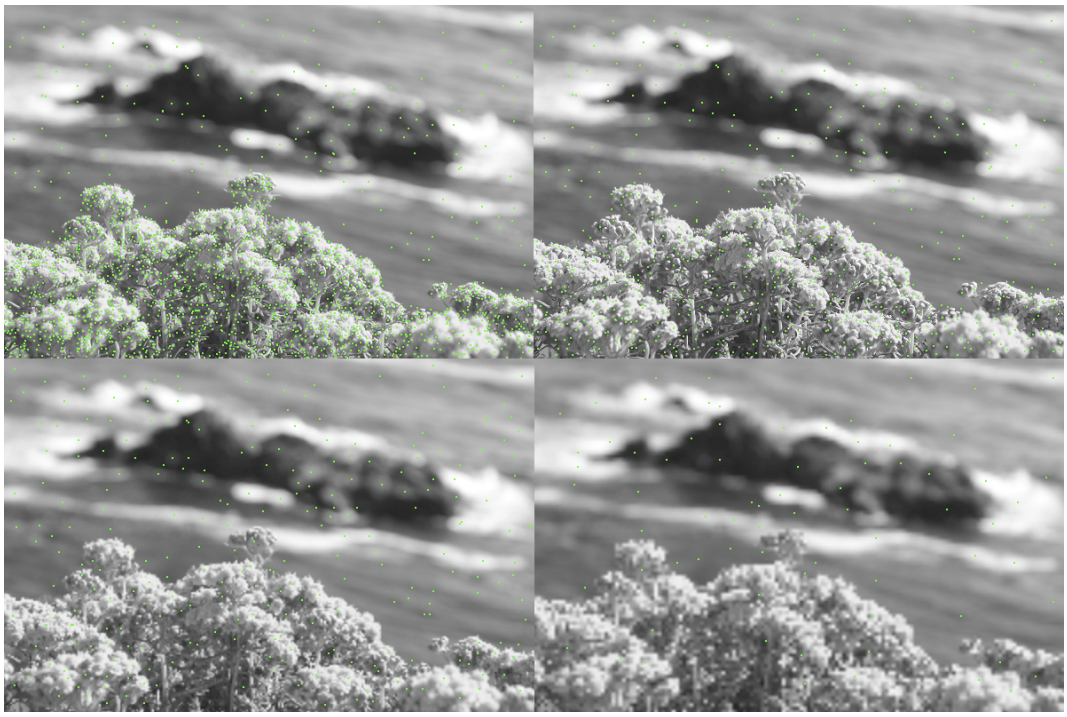

**Figure 4.7: SIFT features with different starting octaves (Photo courtesy of Jonathan Diehl):** Increasing the starting octave (-1 to 2) first only has an impact on detail features in the foreground while the background is not affected, yielding a better ratio between foreground and background object in higher octaves

# Chapter 5

# Object Tracking

*"How did you do it?"*
*"I dropped a logic bomb through the trapdoor."*

*—Hugh Jackman in Password: Swordfish*

we chose to develop a point tracking algorithm based on point detection to track objects. The tracker learns and continuously adapts an object model based on detected points. The model is not dependent on a specific detector, but can work with any point detection and matching algorithm. In our implementation we decided to use SIFT because of its well known stability and the existence of several implementations, including implementations running on the GPU [Wu, Sinha et al., 2006]. Additionally, we use CAMShift on color histograms as a second tracking algorithm to augment the tracker in situations where the developed point model has problems or point matching fails. In the following, we present our reasons for abandoning optical flow as a basis for tracking, and introduce the new algorithm in detail.

## 5.1 Why not Optical Flow?

Advantages of optical flow

Optical flow has proven to be a suitable basis for finding object trajectories. DRAGON as well as *DimP* use flow based approaches to track objects. It allows for simple

trajectory creation that does not rely on specific object attributes. This is one of the main requirement for direct manipulation as discussed in chapter 3. DRAGON'S dense flow algorithm can be used to track even tiny objects and both algorithms do not rely on rigidity to work. Tracking does not require more information than the one point the user clicked on. As *DimP* shows, computation time can be reduced by estimating flow instead of computing dense flow fields and the SIFT features required can be computed faster with implementations running on the GPU [Sinha et al., 2006]. Also, specialized algorithms exist that can compute dense flow fields more efficiently. Complex solvers, such as the multi-grid solvers presented by Bruhn and Weickert [2005] and Bruhn et al. [2005], can compute up to 18 accurate flow fields per minute for a frame size of 316x252 on a 3.06 GHz Pentium machine.

Disadvantages of optical flow

The main problem of optical flow tracking arises from the lack of object awareness. Simple flow based object tracking as employed in *DimP* and DRAGON has to rely on the assumption that the motion vectors between every frame pair are correct. Unfortunately, these motion vectors do not necessarily need to be correct mathematically, they need to be correct for the user. Assume the case of a perfect optical flow algorithm that can map any pixel on the screen to its respective position in next frame, defined by motion of the corresponding 3D object. This is still not enough, because the user's notion of correctness changes with the object he is interested in. Especially in the case of occlusions, the difference between mathematically correct and correct for the user are drastic. In figure 5.1, the user will be interested in the movement of the person. If the tracker simply follows pixels, the trajectory will stop on the tree, which is correct if pixels are considered, but incorrect for the user. This implies that object awareness of the tracking algorithm is a crucial point for direct manipulation. Although the underlying optical flow computation usually address objects implicitly with special constraints, objects are not explicitly dealt with in the algorithm.

The current tracking techniques built on top of optical flow have no notion of objects. Thus, the algorithms cannot determine when occlusions occur and even small partial oc-

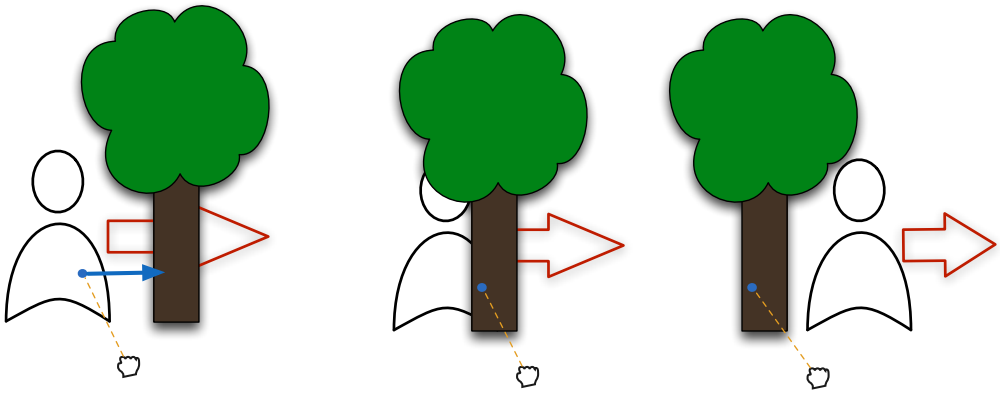

**Figure 5.1: Occlusions and optical flow:** The optical flow vectors (blue) on the tree are the null vector, because there is no motion of the tree. Thus, the person cannot be followed by simple flow tracking while occluded.

clusions usually produce tracking errors 5.1.

Neither DRAGON nor *DimP* are able to detect occlusions and recover from them. If an algorithm recovers from occlusions it is due to the underlying flow algorithm. In case the tracked object is large enough, fast enough, and the occluding object small, the smoothness constraint in DRAGON'S flow algorithm produces motion vectors that help to recover from the occlusion (Figure 3.2). The smoothness constraint expects spatially close pixels to move coherently, which is based on the assumption that spatially close pixels originate from the same object in the 3D scene. This can affect flow vectors close to the border of objects incorrectly, because the assumption does not hold near the edge. In the streetlight example 5.2, there is no apparent movement of the streetlight and thus, all vectors originating on the streetlight are supposed to be the null vector. The fact that the tracker follows the car if the user clicks on the street light shows that this is not the case. In this case the incorrectness helps the interaction, since the user will usually aim to interact with the car. Problems arise when it unexpectedly does not work (Figure 6.21), or the user actually wanted to interact with the smaller object.

The smoothness constraint produces incorrect vectors

Occlusions by small objects are also less problematic for *DimP*, since small object have less features that may occur in two adjacent frames impacting the flow field interpola-

**Figure 5.2: Resolving occlusion in DRAGON:** When clicking on the streetlight a trajectory fitting the car is computed. This helps to resolve occlusions, when a motion vector of the car ends on the street light.

Occlusions are only handled as a side effect

tion. Again, this is not intended, because tracking of these small objects is impossible. In both cases, the technique was never specifically designed to handle occlusions and the effect on occlusion recovery is rather a side effect, which only occurs when the occluding object is sufficiently small. Since occlusions are very common in video scenes, direct manipulation requires an algorithm that can deal with this problem explicitly.

Occlusion aware optical flow

Several approaches extending optical flow algorithms with occlusion awareness exist [Ince and Konrad, 2008]. These approaches are tailored to extrapolate optical flow in disappearing regions — i.e. regions occluded in the second frame. Extrapolation is necessary to compute vectors describing the behavior of the disappearing pixels. Independent of occlusions, this is a desirable quality for direct manipulation, since trajectories are not supposed to end before an object disappears. Nevertheless, this alone is not sufficient to determine pixel position over multiple occluded frames. To ascertain the correct position, the algorithm

needs some way to decide whether the tracked pixel is visible or hidden. In the first case, it can use the flow between adjacent frames to continue tracking and in the second case, the "hidden" flow has to be estimated or computed in a different way.

Occlusion handling requires an object model

Handling occlusions requires us to have some model of the object. This model can be based on any distinctive feature like color, gradients, points, flow vectors or a combination. Instead of implementing an object model on top of optical flow to rectify flow vectors not fitting the current situation, we propose to employ a tracking algorithm that bases its tracking decision directly on the object model.

## 5.2 Point Tracking

We propose to introduce an object model that is suitable for tracking as well as detecting and handling occlusions. The object model is built of points which are spatially related to an initial point — i.e. the point the user clicked on. This is the point that is supposed to be followed across frames to build the trajectory. Several points represent the object. These points may have occurred over the course of multiple frames and change according to the current lighting conditions and object form. In every frame, the algorithm extracts existing points and then finds the points that fit the model best. The best points are used to determine the current object position. Newly occurring points are evaluated in terms of whether they fit the model and, in case they do, are incorporated into the model. In addition to points, recent movement is part of the model. Thus, if not enough of the extracted points fit the model — for example due to motion blur or occlusions — movement can still be estimated based on the assumption that it changes slowly over the course of several frames. As soon as extracted points fit the model again, the algorithm proceeds as before. Contrasting to the flow implementations, we use several frames as reference. In this way, points shortly lost by the point detector, for example because of a person turning their head or an occlusion, can be used again as soon as they resurface. Moreover, it increases tracking stability, should the point

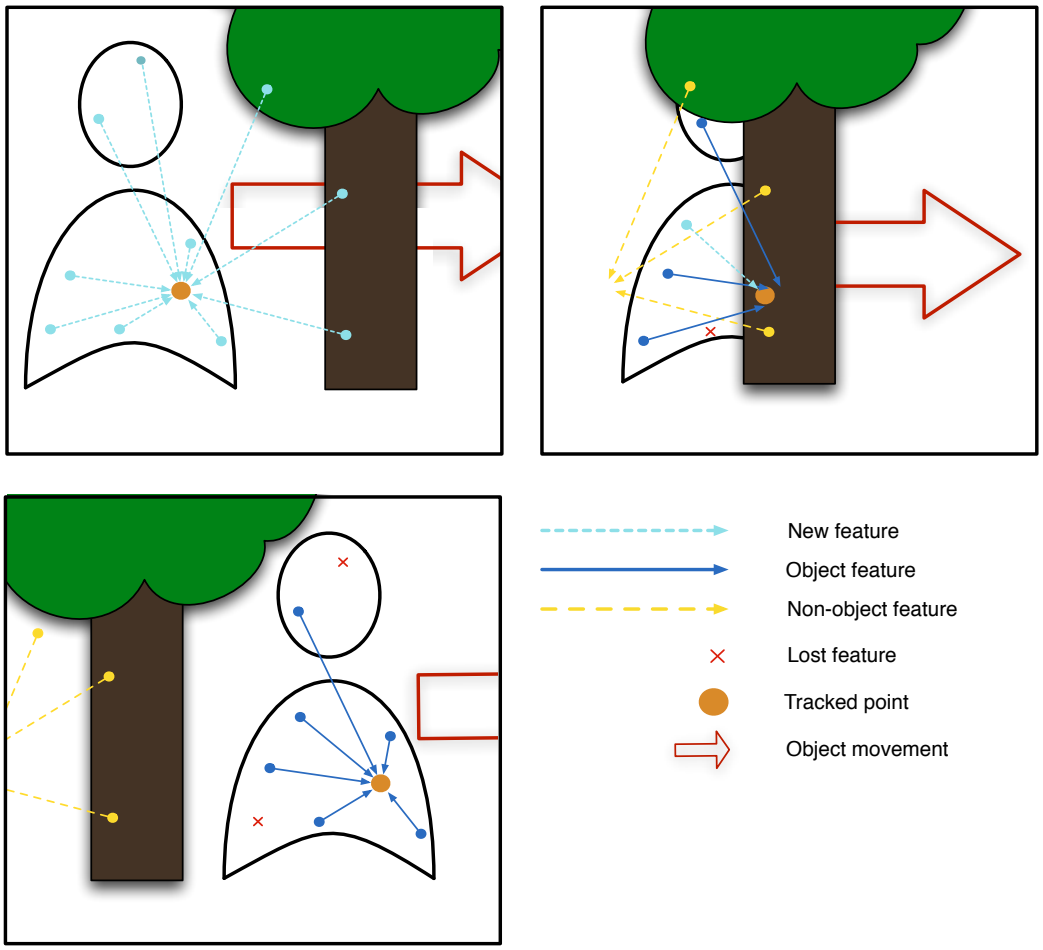


**Figure 5.3: Visualization of the developed point tracker:** Top left: The tracker learns features in the first frame. Top right: features are classified of belonging to background or foreground. Bottom: Older feature not found in the previous frames are used as reference

detector fail to find points in every frame. Using multiple frames is realized by trying to to find the object in a frame, instead of determining its movement from one frame to the other. All parameters mentioned here were chosen by preliminary testing on several videos, non of which were used for the evaluation.

The tracking idea can be visualized as in figure 5.3 and consists of the following steps:

1. Load the current frame and detect points
2. Match new points to the points contained in the model and candidate points
3. Remove matches that fit the background motion
4. Predict the position of the interest point from each match individually
5. Cluster the matches based on the predicted positions
6. Evaluate each cluster in terms of distance to the interest point and usage of the points in previous tracking steps
7. Update the interest point position based on the barycenter of the best cluster
8. Update the object size based on the points used to forming the cluster
9. Evaluate whether each match fit the determined model
10. Devalue model points that did not fit the object movement or could not be found
11. Add newly found points in the vicinity as candidates for object points
12. If the object position could be determined, repeat with next frame
13. In case the object could no be found for several frames, stop

### 5.2.1 Object Detection

Segmentation by differing movement

Without any knowledge about distinctive features, it is impossible to distinguish foreground and background. Thus, before the model can be built, the object needs to be set apart from the background. We decided to use the object movement to distinguish it from the background. We considered two heuristics of classifying features as object or non-object features.

- Features close to the interest point that are moving according to a common movement model
- Features surrounding the interest point moving according to a common movement model

The first approach can have false positive responses when another object, close to the intended object, shows a better fitting model. Approach two discourages false positives, because now the other object(s) would have to completely surround the correct object. Unfortunately, false negatives can occur when the user clicked close to the object border, where surrounding the interest point with feature points is impossible. Also, depending on the applied point detector, the object might not show features in all parts. After preliminary testing with SIFT, we decided to use the first way because SIFT is a blob detector and discourages points at the border of objects. The other heuristic would require the user to click at points surrounded by non-zero small enough gradients (when using SIFT), which is incomprehensible to the normal user. Figure 5.4 shows an example where a car could not be tracked when clicking on the hood. The relation "farther away from the other object" im-

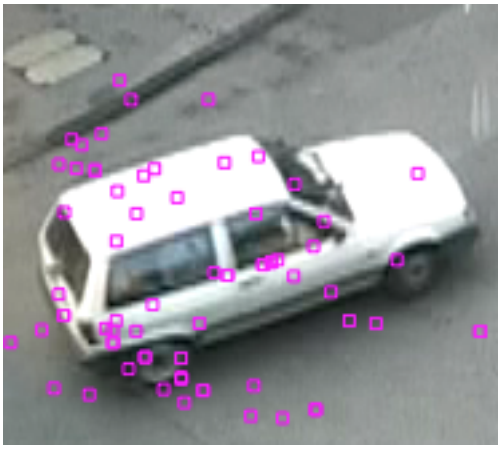

**Figure 5.4: Feature distribution with SIFT:** The car shows only a single feature on the hood. So requiring surrounding features will fail when the user clicks to close to the border or this one point cannot be found in the next frame.

plied by the first technique is easier to grasp. Another point

detector that emphasizes edges could work better with the second technique. In the end, the best technique has to be decided by user tests.

### 5.2.2 Movement Representation

To determine the next point of the trajectory, some kind of movement model has to be determined from model points and applied to the interest point. We considered two movement models for direct manipulation; an affine model (see chapter 4) and a translational model. Translational models only represent translative movement and cannot represent movement induced by scaling or rotation. This does not mean that they cannot take those kinds of movement into account during computation. Affine models can also represent these more complex movements, but they are also harder to compute. We require at least three points to compute an affine model. This is a strong constraint for tracking, making it especially hard to track small objects. An affine model would also require some kind of model normalization to allow for estimation from different reference frames. For example, a detected rotation would have to be applied to all model points to keep the estimation stable. Thus, an error in estimation will always affect these normalized coordinates.
Translational models only require a single feature but are less accurate when it comes to more complicated movement. Since a rotation or a deformation have different effects on different parts of an object, this produces errors when the interest point does not coincide with the point used for estimation. Rigidity is not a given prerequisite, but we can assume that objects do not change shape rapidly from one frame to the other. If they do, SIFT, or most other point detectors, will not be able to match points reliably anyways because of motion blur. Thus, the error introduced by assuming rigidity will usually be small. Also, more points can be employed to estimate the position of the interest point to reduce such errors. It would preferable to have an object model that has information about every part of the object. Such a model would allow for new interactions incorporating zoom and rotation as well. Unfortunately, initial testing showed that the constraints reduce

tracking stability and especially small objects show unstable trajectories.The user tests mentioned in chapter 3 indicate that accuracy is not one of the main concerns. We believe that a translational model that considers the possibility of zoom and rotation will be sufficient for the current interaction.

### 5.2.3 Position Determination

Step 11 — Adding new points

To track an object, any point in radius $r$ around the interest point $I$ is considered as a possible object point. For every considered point $P_j$ the vector

$$\vec{v}_j = P_j - I$$

defines the point's relation with the interest point. This vector is computed when a point is found the first time (Step 11). Each point has a score for tracking importance $s(P)$, representing how reliably it could be used for tracking in the past. Initially it is set to 1.

Step 2 — Removal of background points

Figure 5.5 visualizes the tracking process. After computing points for a frame (Figure 5.5 a), each model point is tested for correspondence with the new points. All points that could be paired, and do not match the camera motion are subsequently tested for a common model (Figure 5.5 b and c). A point is compliant to the camera motion when

$$||H_{\text{Camera}} \cdot P_{\text{Origin}} - P_{\text{Match}}||_2 < \epsilon$$

where $||\cdot||_2$ is the euclidian norm and $H_{\text{Camera}}$ is the homography defining the camera motion from the frame the point was found in originally to the current frame. $\epsilon$ compensates for the error in the estimated homographies. If camera motion compensation is disabled, only points that do not move at all are removed. A point that was already part of the model at some point, is never removed in this step. This ensure the ability of tracking objects stopping in the course of a scene.

Step 4 — Position estimation

In step 4, each point $P_j$ is used to estimate the new position of the interest point $I'$ by adding the vector $\vec{v}_j$ to the new

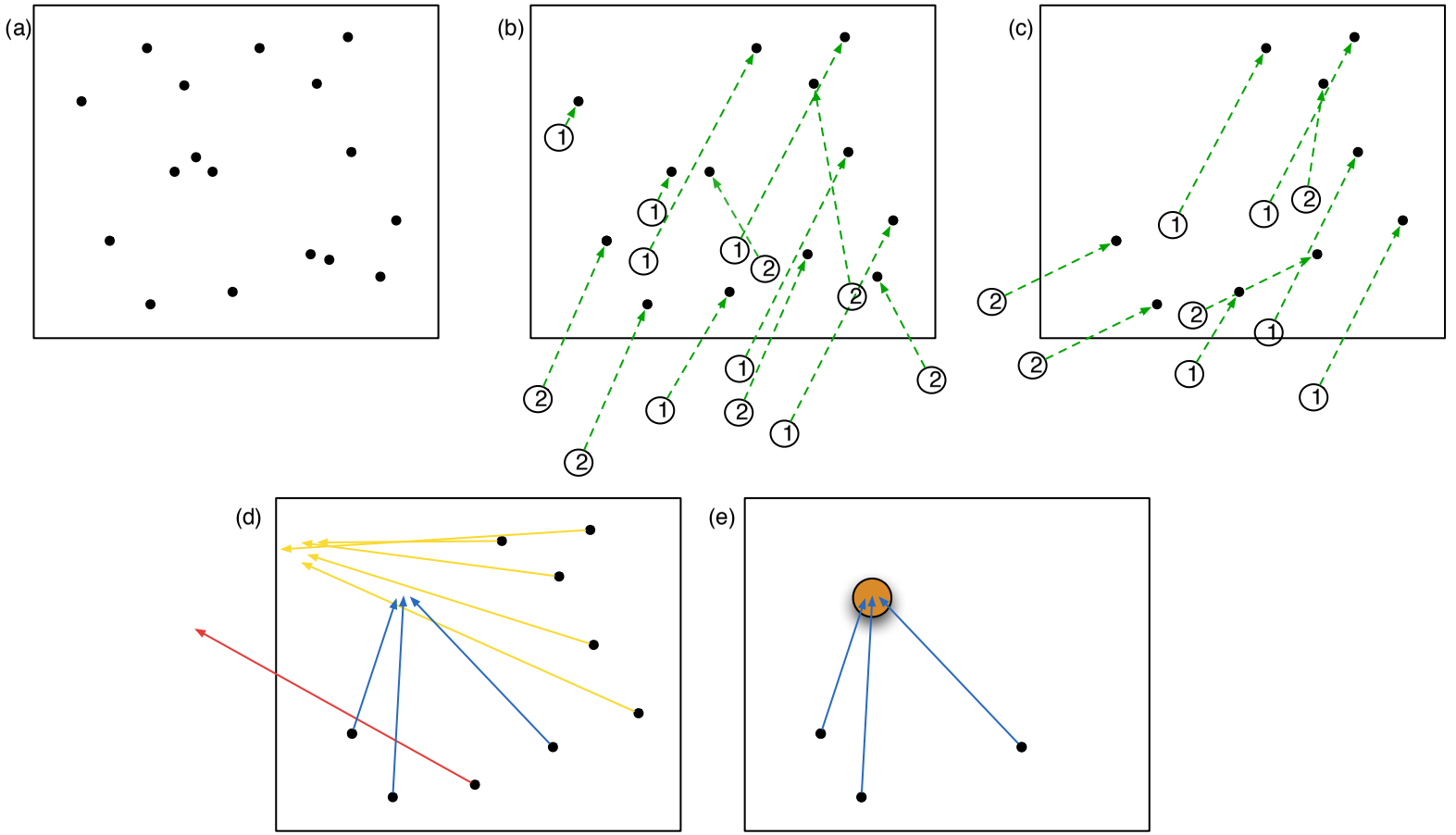


**Figure 5.5: Visualization of the tracking process:** (a) point detection (b) point matching: the number represents the age in the model (c) camera motion compensation and removal of background features (d) interest point estimation and clustering (e) the best cluster and the estimated interest point

position $P'_j$ (Figure 5.5 d).

$$I'_j = P'_j + \vec{v}_j$$

Step 5 — Clustering

This yields a set of possible positions for the new interest point, which can be visualized like in figure 5.5 d. To estimate the correct position, this set is divided into clusters . These clusters represent estimations that follow a common model, that is to say the cluster. Thus, the clustering algorithm has to be able to deal with

- an unknown number of clusters, caused by multiple objects in the same area.
- an unknown number of outliers, caused by points on the background that could not be discarded
- differing cluster shapes, caused by non-rigidity, rotations and zoom

Clustering with DBSCAN

Clustering is performed by a variant of the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) clustering algorithm. DBSCAN is a clustering algorithm that does not require a predefined number of clusters, it can handle outliers, and clusters do not have a predetermined shape. DBSCAN clusters points according to the following rules

1. If all points were classified, stop
2. Get next unclassified point $P_u$
3. Get all points $C$ closer to $P_u$ than distance $d$
4. If $|C|$ is smaller than minimum cluster size $k$, classify $P_u$ as noise; go to 1.
5. Mark point $P_u$ as belonging to cluster $C$
6. If all points in $C$ processed; go to 1.
7. Get next point $P_c$ from $C$
8. Mark point $P_c$ as belonging to cluster $C$

9. Get all points $N$ closer to $P_c$ than distance $d$
10. If $|N|$ is smaller than minimum cluster size $k$; go to 6.
11. Add all points in $N$ to $C$; go to 6.

In our implementation, we chose $d = 4$ and $k = 3$. The next step is to assess the emerging clusters $C$ by how well they fit the interest point , and how they are structured. The estimated interest point of a cluster $C$ is defined as its barycenter $I_C$. After the model initialization phase in the first three frames, at least $l$ previously used points are required, or the cluster is discarded. The idea is to get rid of clusters that have no relation to the model. Our preliminary tests showed that $l = 1$ is unfortunately to sensitive to false points in the model, and we had to choose $l = 2$, which makes tracking hard for small objects with less stable points.

Step 6 — Cluster evaluation

For each point the distance from the original point to the respective interest point $I_C$ is computed.

$$I = ||I_C - P_j||_2$$

The average over all values in a cluster $\bar{i_C}$, as well as the average tracking importance $\bar{s_C}$ and number of points $n_C$ in the cluster are normalized according to the maximum values in all clusters.

$$\hat{i_C} = \frac{\bar{i_C}}{\max_{\mathrm{C}} \bar{i_C}}$$

$$\hat{s_C} = \frac{\bar{s_C}}{\max_{\mathrm{C}} \bar{s_C}}$$

$$\hat{n_C} = \frac{\bar{n_C}}{\max_{\mathrm{C}} \bar{n_C}}$$

These values are weighted according to $\alpha$, $\beta$ and $\gamma$ with

$$\alpha + \beta + \gamma = 1$$

Finally, this yields a cluster quality $q_C$.

$$q_C = \alpha \cdot \hat{s_C} + \beta \cdot \hat{i_C} + \gamma \cdot \hat{n_C} \tag{5.1}$$

We chose to combine all values in a single quality value instead of defining a disjoint hierarchy like for example the lexicographical order. The reason is that a single value is not meaningful enough. For example, consider a video of a launching space shuttle where the user wants to manipulate the shuttle. As long as the external booster is attached, features on the booster are as reliable as on the rocket. The moment the booster is separated from the rocket, its features become incorrect for the shuttle path. If these features were for some reason a little bit more stable than the ones on the shuttle, only considering stability would lead to incorrect trajectories; namely the booster trajectory. Combining the values avoids this effect. Now, the smaller distance to the interest point compensates for less stable tracking. Tracking importance ($\alpha \cdot \hat{s_C}$) and distance from the interest point ($\beta \cdot \hat{i_C}$) are the most important properties. A high tracking importance relates to the tracking reliability of the points in the cluster. Distance from the interest point ($\beta \cdot \hat{i_C}$) can discriminate between several clusters with good tracking importance. The idea is, that a cluster that originates from points close to the interest point is better than a cluster that originates from points close to the border of the object. Secondary to the shuttle example, this is interesting in the case of nonrigid objects. Several parts of the object that could be tracked together before might now be independent of each other. The part most important to the user is probably the one closest to the interest point. Cluster size ($\gamma \cdot \hat{n_C}$) is the least important property because it can easily be dominated by larger objects moving in the background or next to the tracked object. With this reasoning we propose to use

$$\alpha \approx \beta >> \gamma$$

Our implementation uses $\alpha = 0.45$, $\beta = 0.55$ and $\gamma = 0$.

Steps 7 and 8 — New interest point and object size

The barycenter of the cluster with the highest quality $q_C$ defines the new interest point $I'$. The smallest rectangle containing all feature points defines the current object size. To accommodate for missing features because of non-rigidity and occlusions, a maximum size change of 25% is allowed.

A special case has to be considered when the object is not moving. This means, that all model points as well as background points conform to the object movement which is in this case equivalent to the background movement. To avoid classifying background points as stable model points, new model points may only be learned when the object is moving. Since a static object does not show a lot of feature changes this should not reduce tracking stability. This makes the tracker dependent on correct camera motion estimation. In the case camera motion was identified incorrectly, and background points are classified as parts of the object, $\beta$ should be large enough to compensate for the effect and prioritizes points close to the interest point as soon as movement starts again.

### 5.2.4 Model Update

Step 9 and 10 — Point evaluation

The vector $\vec{m} = I' - I$ from the old to the new interest point defines the object translation for the respective frames. The model is updated according to this vector. Each point receives a new importance value $s_t(P)$ for frame $t$ which we defined as

$$s_t(P) = \begin{cases} s_{t-1}(P) + 1 & P \in C \\ s_{t-1}(P) - 1 & P \in M \setminus C \\ s_{t-1}(P) - 0.2 & \text{otherwise} \end{cases} \tag{5.2}$$

where C is the set of points used to define the vector $\vec{m}$ and $M$ is the set of points that could be matched in the current frame $t$. In this way, a point that is not found in every frame, but regularly at least every second frame is still classified as stable. Points that do not fit the model are not removed immediately to accommodate for detection outliers and incorrect clusterings. A point is removed from the model or list of candidates when s(P) becomes smaller than or equal to zero. Since the initial value was one, points on different objects are constantly considered as model points but are removed immediately since they do not fit the model. In this way, false negatives are also corrected very quickly since it takes only three frames until they are considered again.

## 5.3 Color Tracking

One of the main problems of point detectors, and SIFT in particular, is the dependence on image gradients. Unfortunately, gradients are very susceptible to motion blur and may change rapidly if an object starts moving. Motion blur has a direct effect on the definition of edges and thus gradients defining orientation and descriptors in SIFT. Although SIFT itself employs gaussian blurs, a motion blur has different effects than a gaussian blur. Especially the fact that motion blur only occurs along the movement axis has a very different impact on gradients than gaussian blur. Figure 5.6 shows an example. To enhance the tracker to be able to

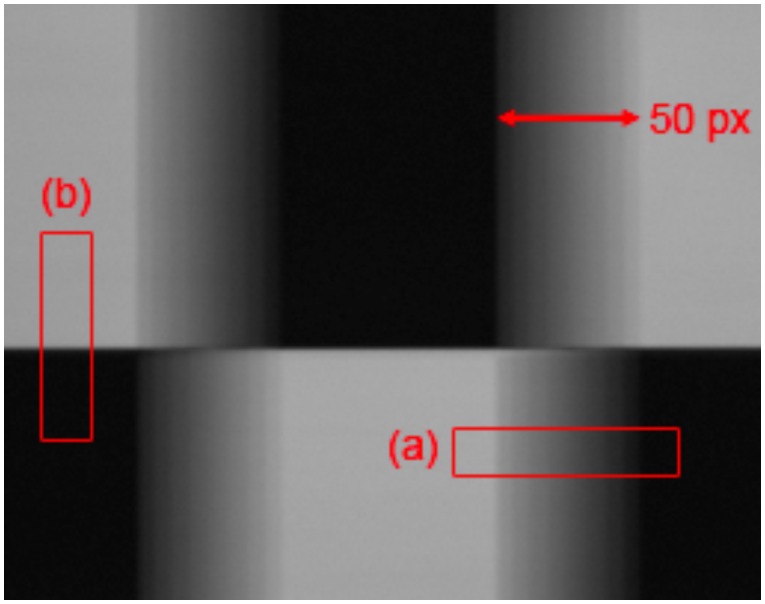


**Figure 5.6: Effects of motion blur [Schuon and Diepold, 2006]:** The underlying structure is a regular black and white grid. Gradients in blur direction (a) are affected while perpendicular gradients stay almost constant (b)

deal with this problem, we propose to use a second, tracking phase, which uses CAMShift (see 3.3.1) based on color histograms. Color is not as susceptible to motion blur, making it a good choice for an additional feature. We chose to implement it based on histograms on the hue component of the HSV color space. The initial kernel of the object has

a size of 10% of image height and width and is centered at the point where the user clicked. Using this heuristic enables us to track objects even if SIFT is never able to detect the object boundaries. Small objects with few features are hard to track by SIFT because multiple features are required for tracking. Figure 5.7 shows an example where CAMShift helps in such a situation.

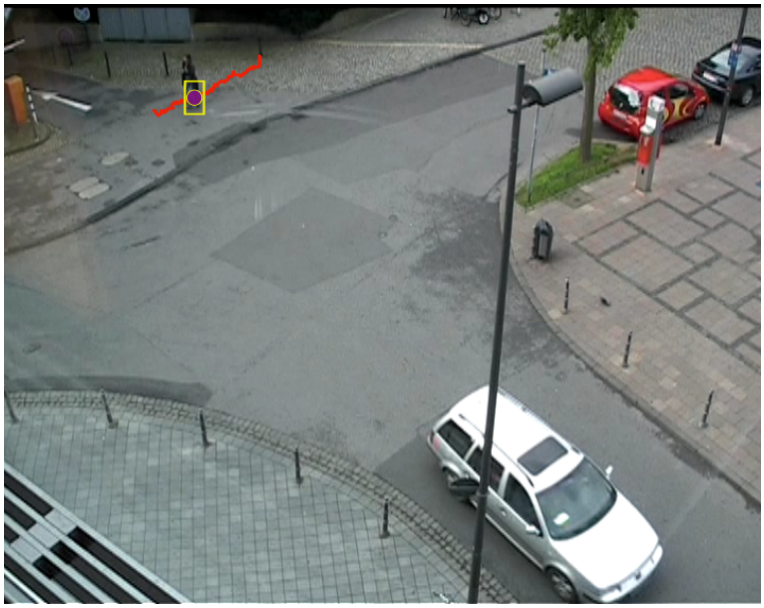

**Figure 5.7: Tracking a small object with CAMShift:** Although the tracked region is only about 20x30 pixels large, CAMShift produces a good trajectory.

### 5.3.1 Model Combination

Combination of point and color tracking

Combining the proposed models requires the detection of errors in one of the models. SIFT can distinguish objects in a scene very accurately. Thus, as long as the point tracker is able to track the object, we chose to trust it. Also, as long as the point tracker is able to track the object, tracking results are used to test correctness of the CAMShift tracker. When the movement of the CAMShift tracker differs from the movement indicated by SIFT, CAMShift is newly initialized to avoid diverging models. If SIFT tracking is not

possible, CAMShift will be trusted as long as the size does not change drastically in between frames. This means a size increase of 33% over five frames. This does not hold for the first five frames after initialization to allow the kernel to grow to the object size. Fast growth often indicates that the tracker has lost the object an started to track colors of the background.

## 5.4 Implementation

We implemented the object tracking in Objective C with Cocoa on Mac OS X 10.5 and incorporated both into DRAGON. For point detection we employ SIFT, which is well known for its stability to changes in object pose and illumination. Loading single frames from a video is done via the Quicktime API. The latter proved to be very slow for random access of video frames, which can take several seconds. Thus, instead of loading single images from the stream, multiple frames are preloaded and cached, allowing Quicktime to make use of intermediate data during the extraction. For the backward estimation, frames are also read forward from the farthest frame on to ensure the use of the optimized forward decoding algorithms.

### 5.4.1 Parallelization

Speed increase by parallelization

Achieving frame rates for real time application with the proposed algorithm requires a high degree of parallelization. For example, SIFT features for the next frame can already be extracted while the camera motion and object model are computed. The parallelization is implemented using the concept of operations, provided by the Cocoa API. Operations are designed to partition a task into smaller subtasks which can be handled by different processors in parallel. Programming with operations only requires the definition of the relation between individual tasks. We segmented camera motion estimation and tracking into the following subtasks

- *Image loading*
  Decoding video can take some time, and is thus done separately.
- *SIFT feature extraction*
  Handles extraction and storage of SIFT features.
- *Homography estimation*
  Uses the points determined by the point extractor to estimate the effects of the camera motion.
- *Point tracking (model update)*
  Tracks the objects by means of the described point tracking algorithm
- *CAMShift tracking*
  Tracks the object with CAMShift based on color histograms
- *Merge of tracking results*
  Combines the result of the two tracking approaches to a single position
- *Success test*
  Tests whether tracking has to be stopped because the object was lost by both algorithms

The relation between these steps are shown in the dependency graph 5.8. Additional steps are possible. For example model features can already be matched, while camera motion is computed. On the other hand, the speed increase from this will be marginal, since the model update is very fast anyways (section 6.3).

### 5.4.2 Combination of Object Tracking and Camera Motion Estimation

In theory, the same image is used to compute the camera motion and the position of the object. Thus, SIFT features have to be extracted only once for each frame. To compute SIFT features, we use the GPU implementation by Wu Wu. The slowest part of this SIFT implementation is downloading feature descriptors from graphics to main memory.

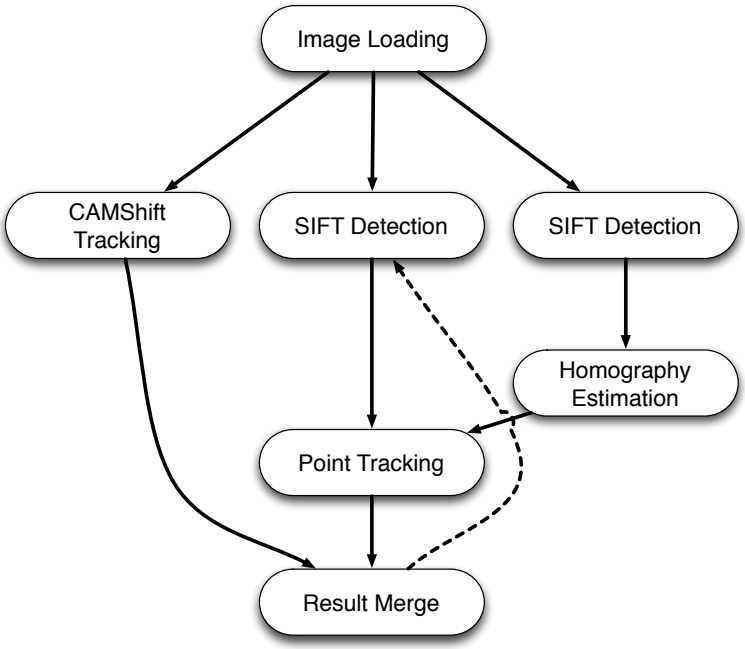


**Figure 5.8: Operation dependency graph:** Relations between the steps of the tracking process

Images are scaled to a resolution of 64x64

Features cannot be downloaded selectively, making it infeasible to compute features for camera motion and object tracking simultaneously. When considering a whole frame, camera motion estimation only requires features on high pyramid levels, while object tracking also makes use of features on lower levels. Objects can be small and thus require more low level features for stable tracking. Downloading all low level features in the areas irrelevant for object tracking would take too long. Instead, the object is cut out, yielding separate images for background and objects. SIFT features are extracted for each of these images. This is still not the optimal way because the image pyramid has to be initialized with the similar data twice and some keypoints are detected in both extractions. To achieve invariance to different object and video sizes, the two images are scaled to a size of 64x64 using resizing with CGImage with high interpolation enabled. The aspect ratio of the window is always left untouched. This has the same effect as starting on a higher pyramid level. Images smaller than that will not be

altered before the detection. This measure reduces feature count, while increasing speed of the SIFT detection step. Especially features in finer structure of an object are lost. So, the extracted features will ensure that the movement of the whole object and not a small detail is preferred. This will usually be of more interest to the user. If the user indicates that she is interested in finer structures, the algorithm could put more weight on lower pyramid levels containing information about details. Lowe [2004] chose the SIFT standard parameters to achieve maximum stability. This is important for our algorithm as well since it allows to download as little features as possible. Sometimes it might be a good idea to increase feature count even if it costs stability. For example to create new trackable features if an object has too few. This can be achieved by using a lower pyramid level, or a different number of levels for each pyramid octave. Unfortunately, the SIFT implementation does not allow to choose these parameters dynamically. These and the aforementioned problems with the SIFT implementation show the necessity of a new implementation tailored for direct manipulation. Images that are used for CAMShift tracking are not scaled down, because CAMShift works very fast, so keeping the information can benefit accuracy.

A more specialized SIFT implementation is required

# Chapter 6

# Evaluation

The developed algorithms were tested on artificially created data as well as on real videos. The artificial data was mainly used to determine limits of the algorithms, whereas video data showed the usefulness in real world circumstances. We decided to not do user test since the new algorithms do not lead to new interactions, but rather to a different realization of the old interaction technique. Especially the actuality that computation takes less time, cannot be evaluated with user tests. In fact, we would expect worse results with the new technique compared to the old DRAGON implementation since users cannot be made aware of the necessary precomputation step. Also, Christian Brockly currently works on several tests concerning the interaction in a second diploma thesis.

User tests

We used twelve videos with varying resolutions and quality to compare the different algorithms. None of the videos were used for preliminary testing during creation of the algorithms and the trajectories and points of origin were chosen beforehand to represent different problem cases. Content of the videos covered simple movement like translation only, more complicated movement like rotation around the y axis, different camera motions and all sorts of occlusion. All videos were taken with a full HD 1080i camera. The videos were deinterlaced with motion prediction in Final Cut Pro. We tested versions with resolutions of 1920x1080, 960x540, and 480x270. Videos were stored un-

compressed to allow using the mov container as well as avi for compatibility with DimP. We did some additional tests with H.264 compressed videos for the new algorithm, which showed no visible difference in the output trajectories. Since most of the tests were redundant, this chapter will only show interesting results. Description of the remaining scenes and trajectories can be found in appendix B.

Unfortunately, we discovered a bug in the GPU SIFT implementation shortly before evaluation. Because of the bug, incorrect descriptors are assigned to to the features. Since we could not find a way to fix it in time, we decided to do all tests with the CPU implementation by Vedaldi [Vedaldi].

## 6.1 Camera Motion Estimation

Experiment creation

The motion estimation algorithm was evaluated using artificial data. This was done mainly because it is hard to create realistic video scenes with a known camera motion. We developed a program that can create the two necessary set of input points for the algorithm, i.e. original and matched points. Original points can be placed manually, randomly or in a predefined shape. A provided homography defines the matched points of the algorithm.Before the matched points are used as input for the camera motion estimation, a given percentage of the points is defined as mismatches, showing random behavior. Furthermore, all matched points are offset by noise vectors with values randomly chosen from a normal distribution with given standard deviation and mean zero. This offset is called position error. In addition to the computation with artificial points, the homography can be computed from two given images with points detected by SIFT.
The test are designed to answer the following questions.

- Is an homography a valid estimation of camera movement for the application of direct manipulation?
- How do mismatches affect the estimation?
- How does position error affect the estimation?

### 6.1.1 Tests on artificial data

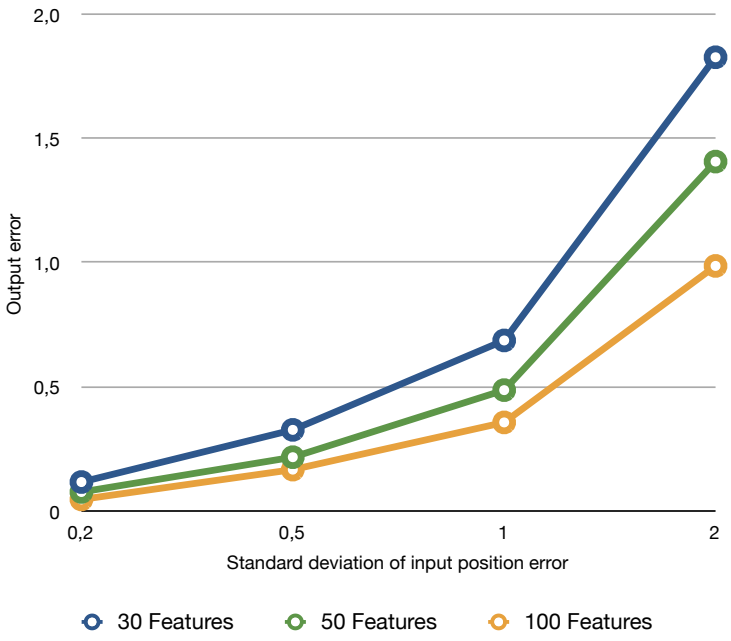


**Figure 6.1: Error of estimation**

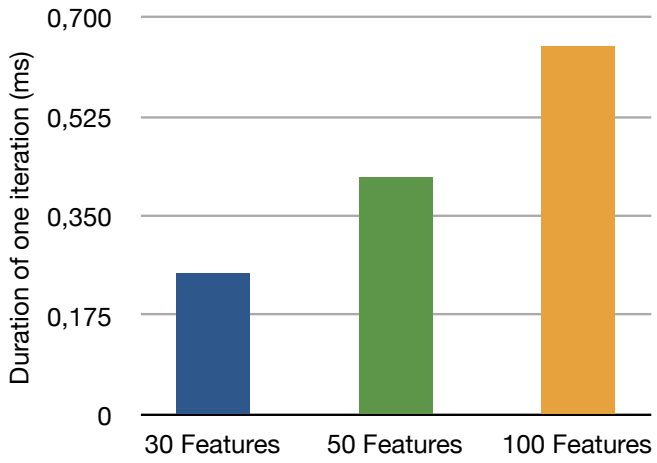


**Figure 6.2: Duration of iterations**

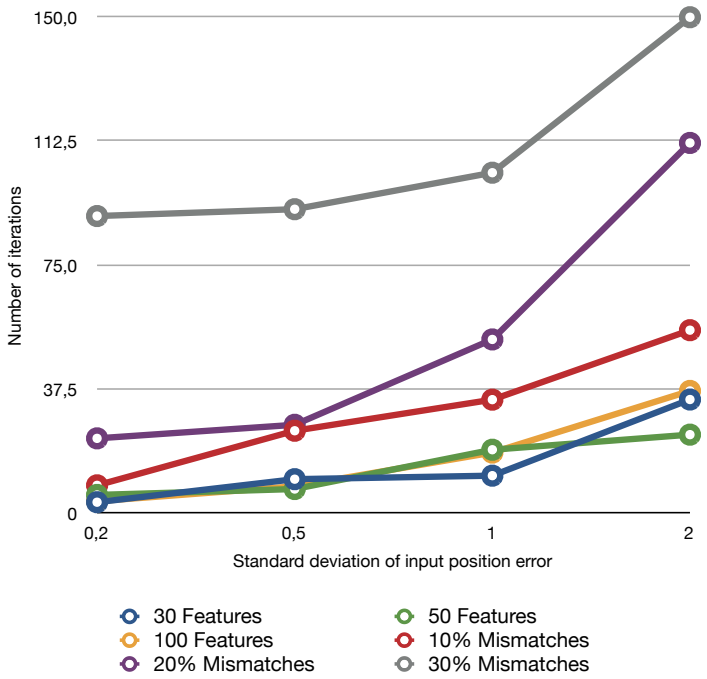


**Figure 6.3: Number of iterations**

The second and third item relate to the point detection algorithm that can be used for camera motion estimation. These tests are done using artificial data. 30 (50, 100) random points are created with position errors of 0.2, 0.5, 1 and 2 pixels. For 30 points, each of the position error values was tested with 10%, 20% and 30% mismatches. Correct matches are defined by a random homography with random values for, rotation ($\in [0, 25]$), translation ($\in [0, 10]$) and scaling ($\in [0.8, 1.2]$). The perspective values ($m_{31}, m_{32}$) are randomly chosen from an interval [0, 0.0001]. The created points are used as input for the homography estimator. The mean of the algebraic error

$$e_a = ||H_{\text{output}} \cdot P - H_{\text{input}} \cdot P||_2$$

for 100 points placed on a regular grid and the number of RANSAC iterations are used as comparative value. Each test is done 50 times. The results show that the error stays relatively small with growing position error. An error of 2 pixels, will be acceptable if it happens only occasionally.


**Figure 6.4:** Stable estimation of a pan (80 frames)

Using more features reduces the error only slightly, but also costs more computation time for a single iteration. Number of iterations increases drastically with mismatched features (or object features) because of RANSAC. The feature number has no significant effect on necessary iterations. A less accurate algorithm than SIFT could work in terms of the resulting homography. Although the necessary iterations increase, in common videos, usually more than 30% of the frame will show foreground objects, which have the same effect as mismatches. For this reason, the number of features used should be kept small to avoid increasing the duration of the estimation process too much.

### 6.1.2 Tests on real videos

The effect of camera motion is not always a homography since the underlying 3D structure is not taken into account. Thus, the second test consisted of taking a whole sequence


**Figure 6.5:** Estimation of a pan, visualized without intermediate frames

of images and use the homographies between frames to visualize the relation between frames of a whole scene. We did one test where we only related the first and the last frame by multiplying the intermediate homographies to compute the relation between these non-adjacent frames. Another test showed every intermediate frame. The difference between these tests is that errors accumulating by multiplying all homographies will be smaller in the second test. If this looks noticeably better, the interface will have to compute and show all intermediate images, even if the interaction dictates a jump over several frames. This would either require a sophisticated video decoding and caching techniques or a different approach for camera motion estimation to accommodate for the real-time application. we tested camera motion estimation on six videos. Representative results can be seen in figures 6.4 — 6.7 and in appendix B. Camera motion estimation works very well on most of the tested scenes, although the images do not represent a single plane. As expected, the individual error between

**Figure 6.6:** Stable estimation of a combined zoom and pan with motion blur (20 frames)

frames is smaller when showing intermediate frames. On the other hand, the example in figure 6.5 shows that the relation between frames 1 and 80 is still clear without intermediates. Camera motion estimation fails in the presence of large foreground objects that dominate the computation. The street in figure 6.7 barely shows any texture, and thus very few features which could be used for a correct estimation.

## 6.2 Tracking

Video annotation

Usefulness of the tracking algorithms in real world situations was determined using data extracted from 14 video scenes. We identified 22 correct trajectories by hand, allowing for comparison of correct and computed trajectories. Trajectories were extracted from the original flow field

**Figure 6.7: Camera motion estimation with large foreground objects (40 frames):** The bus dominates the scene because the background has very few features

Comparative values

implementation of DRAGON, and the implementation described in this thesis,as well as *DimP*. We use two comparative values; the mean of the euclidian distances between correct points and estimated points $\theta_p = ||p_1 - p_2||_2$, as well as the mean of the euclidian norm of the distance between motion vectors between frames $\theta_{\vec{m}} = ||\vec{m}_1 - \vec{m}_2||_2$. The latter gives additional information how stable a tracker follows an object. If the tracker jumps erratically between parts of the correct object, the position error will stay small, while the movement error increases. Quantizing the quality of the trajectories is not easy. The trajectories defined by hand show errors themselves and users do not necessarily devalue trajectories with less accurate positions of single points. This makes the visual quality of trajectories most important. A SIFT trajectory ending early will stay on the last position to compute the remaining values. Trajectories found in different image sizes will only be mentioned To compare videos of different sizes and make comparative values more understandable, the trajectories are mapped

onto a resolution of 160x90. Table 6.1 shows the results of ten of the trajectories.

| Trajectory | DRAGONEYE | | DRAGON | | *DimP* | |
|---|---|---|---|---|---|---|
| | $\theta_p$ | $\theta_{\vec{m}}$ | $\theta_p$ | $\theta_{\vec{m}}$ | $\theta_p$ | $\theta_{\vec{m}}$ |
| Couple | 2.5885 | 1.0251 | 2.5660 | 0.8642 | 13.7102 | 1.1715 |
| Volleyball (Girl) | 1.4539 | 1.1231 | 1.2684 | 1.1644 | 1.1200 | 1.1069 |
| Volleyball (Ball) | 43.1157 | 7.7961 | 30.8003 | 4.7085 | 29.7602 | 4.6057 |
| Shopper | 3.5385 | 0.2013 | 0.9011 | 0.1508 | 10.0483 | 0.1859 |
| Zoom on car | 2.5404 | 0.7217 | - | - | - | - |
| Zoom and blur | 15.0873 | 4.1576 | - | - | - | - |
| Stroller | 11.3459 | 0.2726 | 5.5481 | 0.2503 | 3.5291 | 0.2653 |
| Layup | 13.2528 | 2.9885 | 5.3253 | 1.4019 | 18.0821 | 1.4219 |
| Basketball | 32.4265 | 1.4981 | 70.7380 | 0.9416 | 62.0682 | 0.9924 |
| Pink shirt | 1.4889 | 0.5783 | 6.0879 | 0.5763 | 10.8381 | 0.6034 |
| Backpack | 3.2816 | 0.8167 | - | - | - | - |
| Cyclist | 20.4849 | 1.2835 | - | - | - | - |

**Table 6.1:** Comparative results

### 6.2.1 Translative Motion

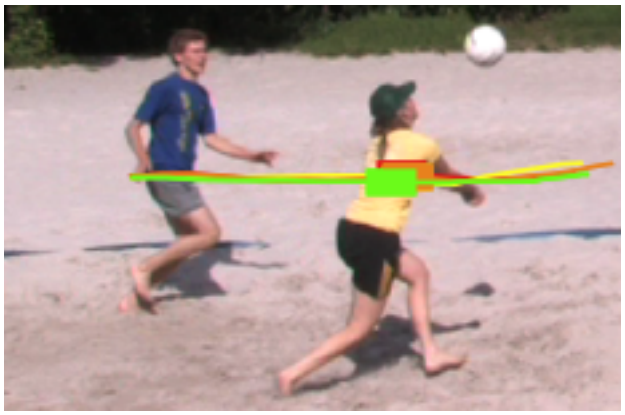

**Figure 6.8: Trajectory 1 — Beach volleyball player:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

We considered 3 trajectories for slow, translative movement. All tracking algorithms produce good results on large enough objects. Only *DimP* looses the couple in scene

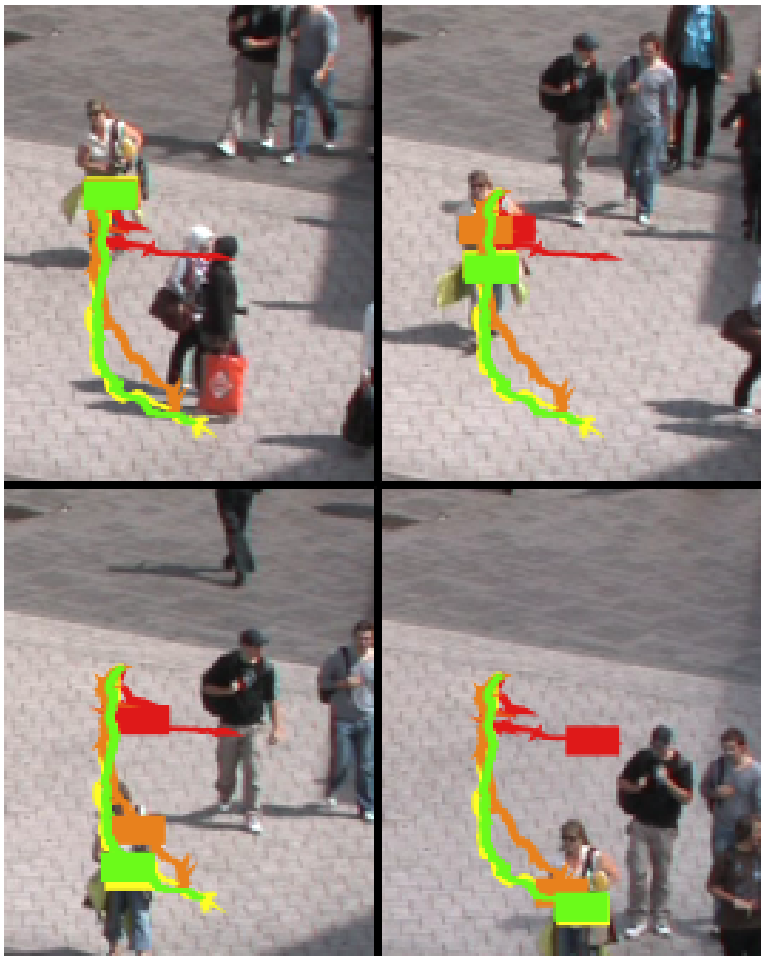

**Figure 6.9: Trajectory 2 — Shopper:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

B.12 after a few frames. The size limits for *DimP* and DRAGONEYE are hard to determine, since texture of an object has a large impact on feature number as well. For example, the girl in figure 6.8 and the woman in figure 6.9 have roughly the same size in pixels. The girl is tracked correctly by both algorithms, while *DimP* cannot find a correct trajectory. Although DRAGONEYE creates a correct trajectory for the woman, its beginning is based on SIFT features on the two men behind her. If they moved in a different direction, she could not be tracked at all. We implemented CAMSHift to help in such situations. Unfortunately, no stable histogram was found until the end. Towards the end,

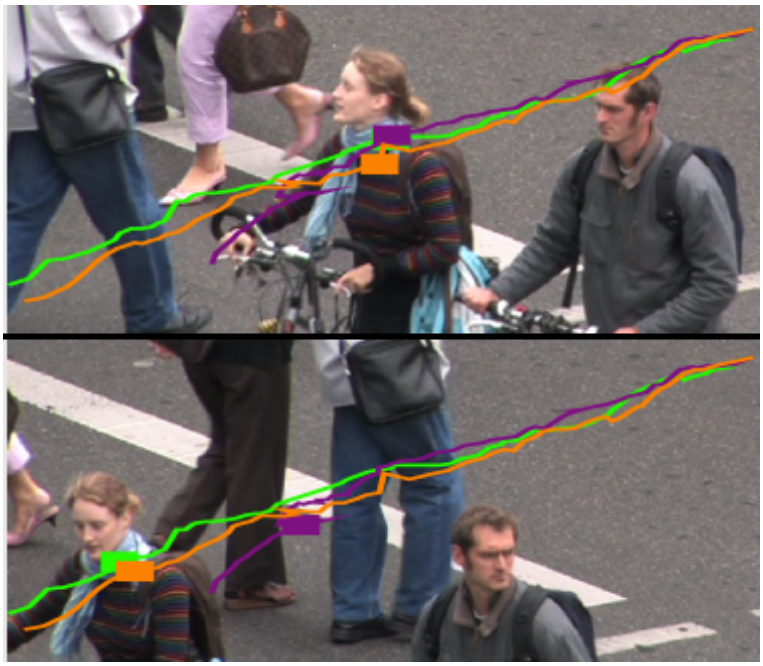

**Figure 6.10: Trajectory 3 — Girl:** Green: correct — Orange: DRAGONEYE — Lilac: DRAGONEYE *(480x270)*

new and stable SIFT features emerge as well. These can be used to track her directly when the two men start to move differently. This shows that the algorithm successfully prioritizes (new) features close to the interest point. Figure 6.10 shows a case where this prioritization fails in the low resolution version of the video. The two cyclist first move coherently, but he slows down at the end. Some features of the man's bag are added for 2 frames, which is already enough to shift the interest point away from the correct features, making correct prioritization impossible. This indicates that the threshold for combined movement should be chosen smaller in lower resolution videos.

### 6.2.2 Complex scenes

Scenes with camera movement, rotation around the y axis, motion blur, and non-rigidity pose bigger problems for the algorithms. Fast and small objects with heavy motion blur, like the volleyball (Figure 6.11) cannot be tracked by any of

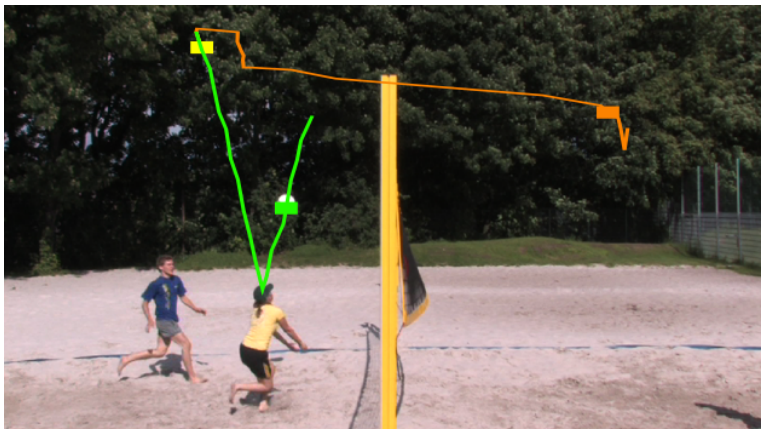

**Figure 6.11: Trajectory 4 — Volleyball:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

**Figure 6.12: Trajectory 5 — Zoom on car:** Green: correct — Orange: DRAGONEYE

the algorithms. Zooms do not pose a problem for DRAGONEYE, as figures 6.12 and 6.13 show. The second video also contains motion blur, which ultimately brings the algorithm to fail. Unfortunately, CAMShift fails as well. A surprising result was produced by the stroller trajectory (Figure 6.16). The scene shows a woman pushing a stroller, turning it around and back toward the camera. The movement is not particularly fast and the stroller is large enough to create sufficient SIFT features. These features cannot stay stable for long because of the rotation, but we expected all tracking algorithms to handle such movement. *DimP* and DRAGON yield good results. DRAGONEYE on the other hand fails after the rotation, when the woman shortly talks to someone before turning back. The reason for this is that background features are shortly affected by the rotation, moving far enough to be taken up into the model. Unfor-

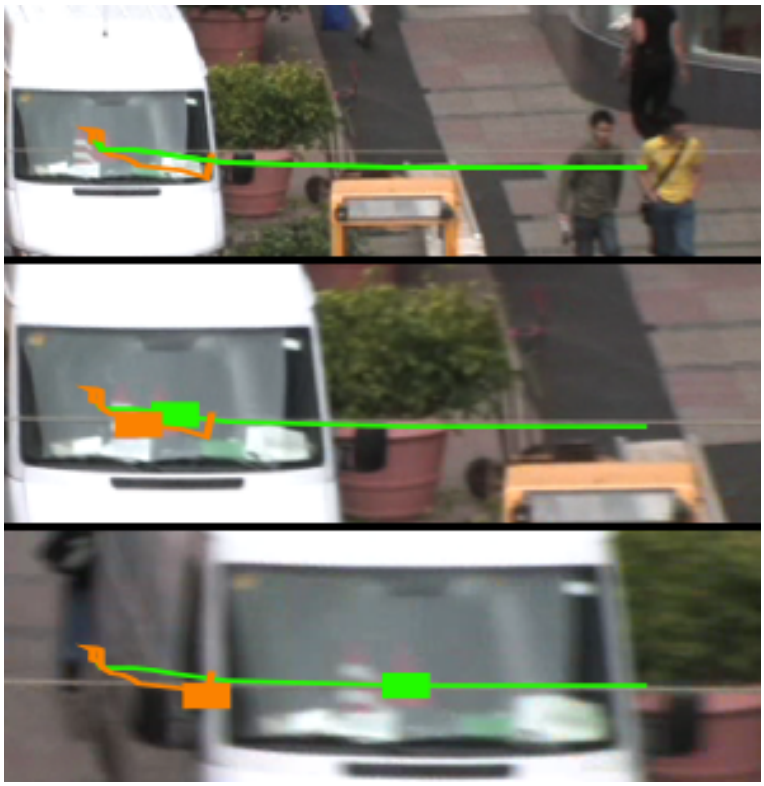

**Figure 6.13: Trajectory 6 — Zoom and motion blur:** Green: correct — Orange: DRAGONEYE

tunately they are not thrown out again since the stroller almost turns around the interest point, which consequently shows only small displacements. Since these background features are stable over several frames they overrule the features emerging when the woman starts to turn back towards the camera.

Layup shot

The next trajectory (Figure 6.14) represents a basketball player doing a layup. The camera follows the player, which produces heavy motion blur towards the end. The player himself shows very non-rigid movement, which should be problematic for SIFT. The optical flow based algorithm performs best in this scene, although the accuracy is lower in comparison to working trajectories in other scenes. SIFT tracking works only for a few frames. CAMShift takes over and performs well until another player in the background enters the tracked region. Both players are tracked simulta-

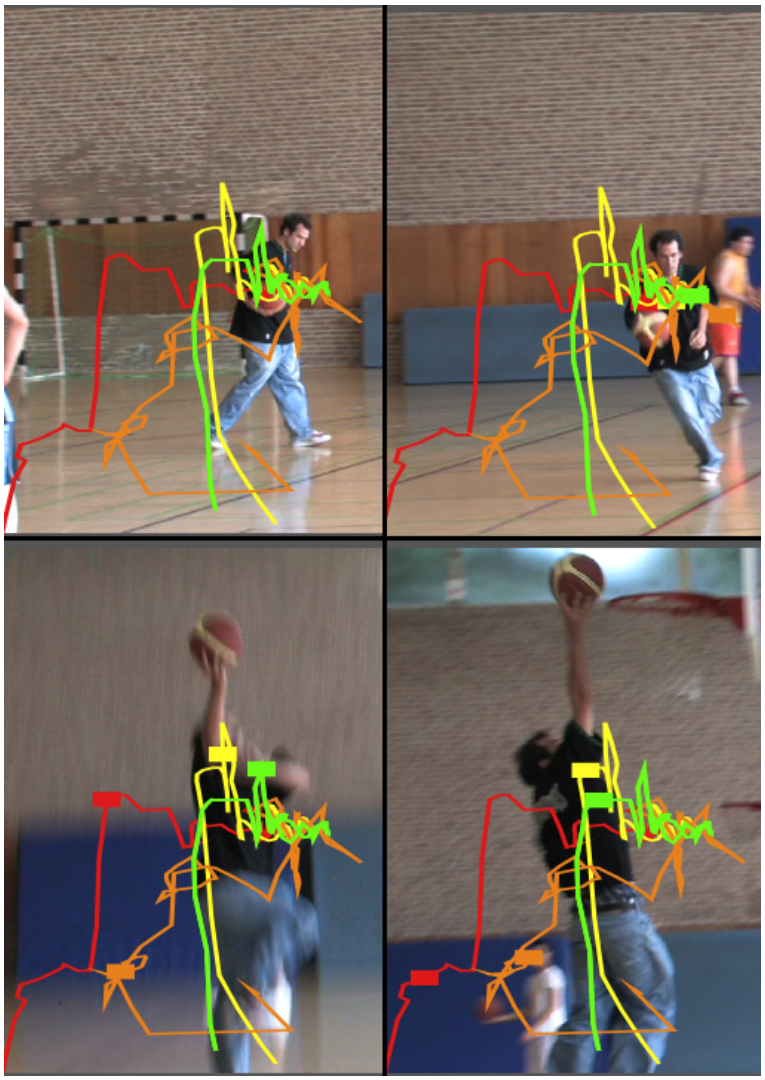

**Figure 6.14: Trajectory 7 — Layup:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

neously afterwards, which produces an incorrect trajectory since they move away from each other. In a higher resolution video, new stable features on the face make tracking feasible for SIFT. This works until the camera moves upward producing heavy motion blur. *DimP* also performs well until the other player comes into view, although the two events are unrelated in this case. The player looses the only stable feature it had for a few frames which suffices to loose track of the region.

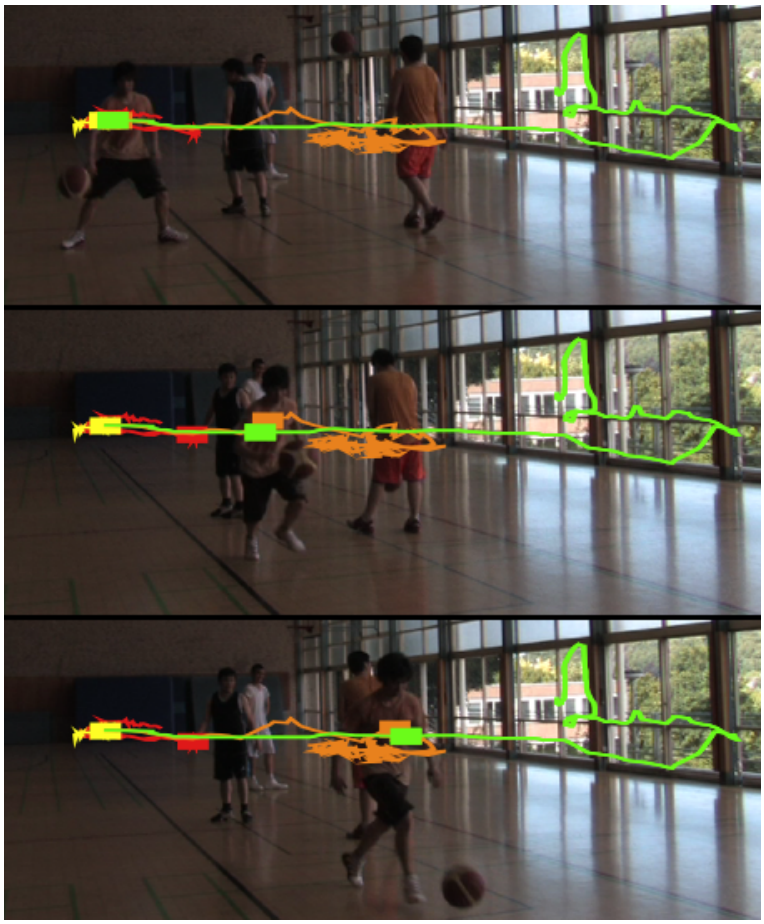

**Figure 6.15: Trajectory 8 — Basketball:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

The trajectory in figure 6.15 follows another basketball player. The scene is badly lit and the player turns around the y axis several times on his way to the basket. Also, the camera vibrates every time the ball hits the ground. Surprisingly, optical flow fails almost immediately without any apparent reason. *DimP* works until the first vibration and DRAGONEYE continues tracking afterwards because of CAMShift. In the high resolution video, CAMShift can follow the player over the whole trajectory.

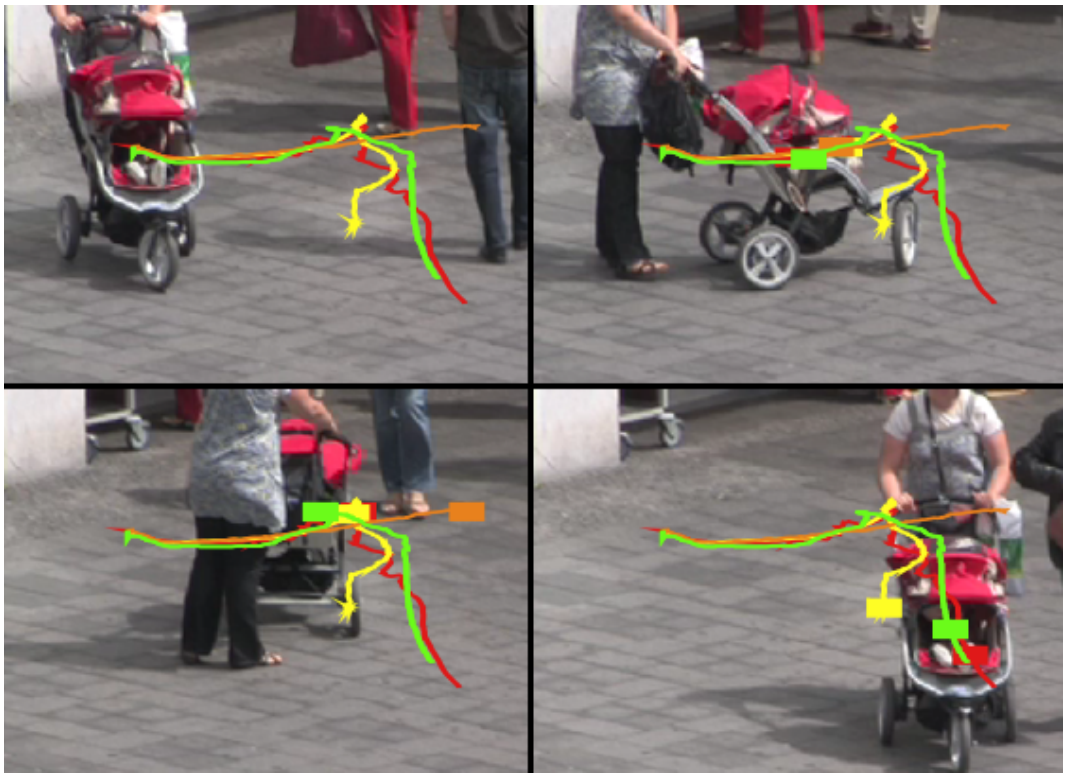

**Figure 6.16: Trajectory 9 — Stroller:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

### 6.2.3 Occlusions

Neither *DimP* nor the optical flow tracking in DRAGON handle occlusions explicitly. Therefore, the following tests mainly show the quality of occlusion handling in DRAGONEYE. We will not show the results of all algorithms here. The only exception is trajectory ?? because the occluding object is only 4 pixels wide. The algorithm has very little problems with partial occlusions. The backpack trajectory (Figure 6.17) shows that it takes less than 5 frames to learn features from the man's back and head (Figure 6.18), which are subsequently used for tracking while the girl's head occludes the backpack. The cyclist in figure 6.19 is almost completely occluded by the flowers on the streetlight, although, there are still features in every frame. The algorithm uses features from old reference frames to track parts of the bike that reemerge on the other side of the street light (Figure 6.20).

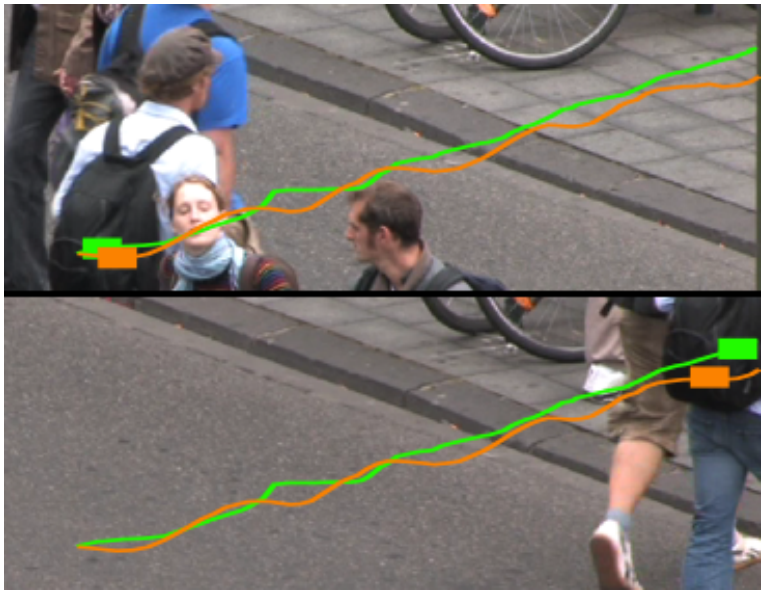

**Figure 6.17: Trajectory 10 — Backpack:** Green: correct — Orange: DRAGONEYE

**Figure 6.18: Backpack — Occlusion Handling:** Two detail views from the backpack scene, the features used for tracking and the cutout used for detecting possible object features

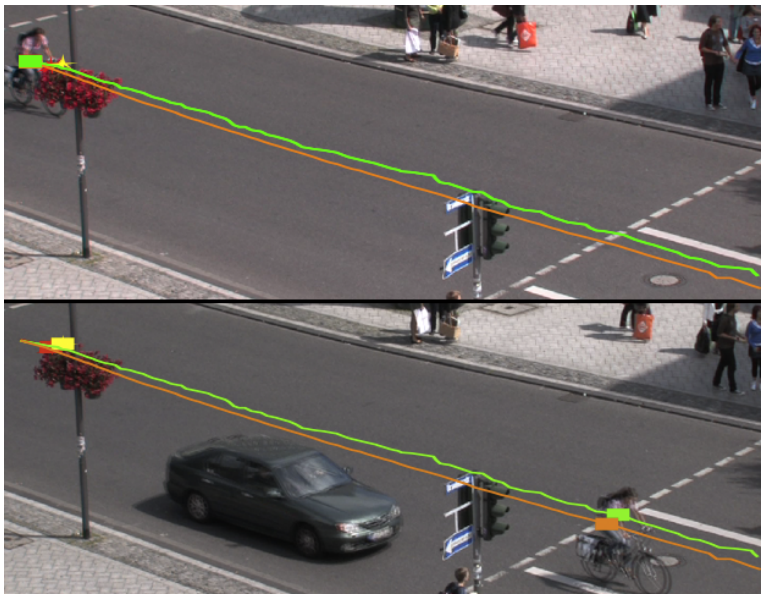

**Figure 6.19: Trajectory 11 — Cyclist:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

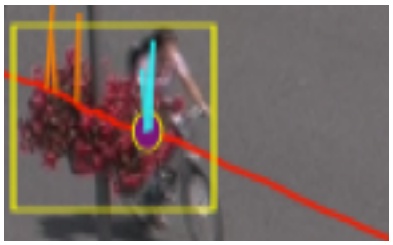

**Figure 6.20: Cyclist — Occlusion handling:** In this frame, the tracker employs old features as reference

Figure 6.21 shows a good example of where the lack of occlusion handling in DRAGON may be surprising for the user. The video shows a woman walking away from the camera, producing slow upward movement in the video. A thin cable, which we did not even notice when we took the video, hangs above the street. In the video, this cable is not more than 4 pixels high, but because of the slow movement, the smoothness constraint cannot create motion vectors crossing the cable. Thus, the trajectory ends at the cable. Because of the slow movement, the SIFT part of DRAGONEYE is not able to track the woman either. CAMShift, on

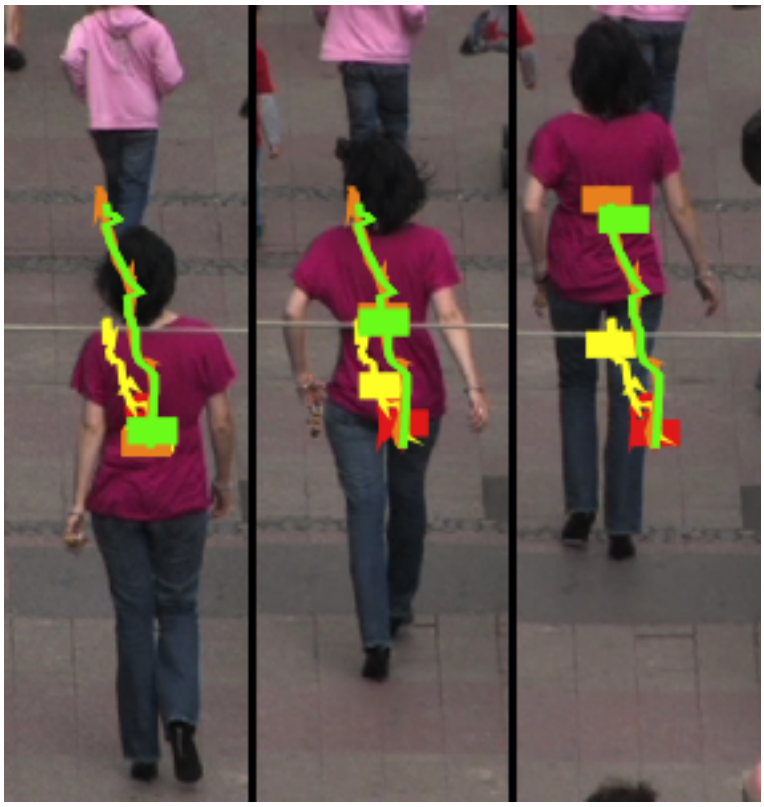

**Figure 6.21: Trajectory 12 — Pink shirt:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

the other hand, can track her nicely, because of the distinctive color of her shirt.

The remaining trajectories did not show any more surprising results. They can be found in appendix B

When it comes to very small objects, such as the famous pigeon[1], can neither be tracked be tracked by *DimP*, nor DRAGONEYE. In this case optical flow outperforms both. In most other cases DRAGON and *DimP* yield very similar results. DRAGONEYE works better in the presence of occlusions, but its SIFT part is a little bit more susceptible to non-rigid objects, which is the price of requiring multiple features for tracking. Although some of the scenes indicate

[1] `http://hci.rwth-aachen.de/download/DRAGON/1057-karrer.mov`

that CAMShift is able to deal with such objects, CAMShift tracking is still to unstable in general. The example with the volley ball shows the need for a new heuristic of when to trust or not trust the tracker, since observing only size changes often does not lead to stopping the tracker.

Different video sizes do not affect tracking stability strongly. From the scenes depicted here, only the shopper and the beach volleyball cannot be tracked in the low resolution version.

## 6.3 Speed

Speed tests were performed on a Mac with a two 2.8 GHz Intel Xeon Quad-core processors and 6 GB RAM. Four cores were used for tracking to leave enough CPU time for additional tasks like the visualization. We measured timing independently for all operations. To validate the benefits of the multithreaded implementation, timing values for a complete model iteration are computed as well. Table 6.2 shows the measured timings and figure 6.22 visualizes the proportions of the individual task duration considering different video resolutions.

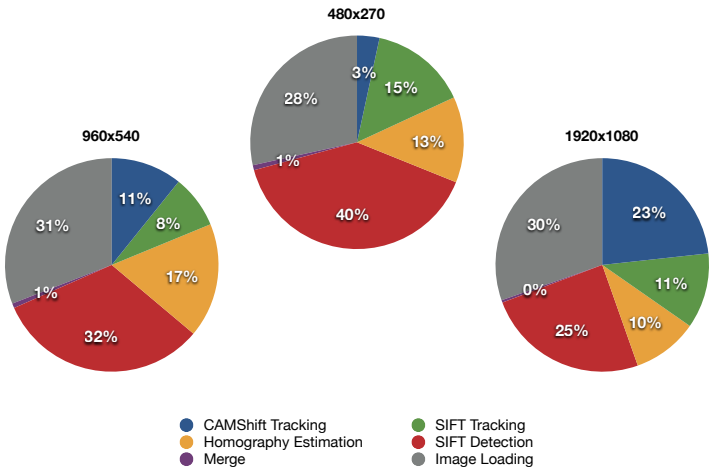


**Figure 6.22: Proportions of task duration**

| Resolution | Operation | Average duration (ms) |
|---|---|---|
| 480 x 270 | Image Loading | 37.4 |
| | CAMShift Tracking | 4.4 |
| | SIFT Detection | 52.1 · 2 |
| | SIFT Tracking | 19.3 |
| | Homography Estimation | 17.1 |
| | Merge | 1 |
| | Complete | 191 |
| 960x540 | Image Loading | 43.1 |
| | CAMShift Tracking | 15 |
| | SIFT Detection | 45 · 2 |
| | SIFT Tracking | 11.1 |
| | Homography Estimation | 24.2 |
| | Merge | 1 |
| | Complete | 224 |
| 1920x1080 | Image Loading | 83.2 |
| | CAMShift Tracking | 63.7 |
| | SIFT Detection | 67.6 · 2 |
| | SIFT Tracking | 31.2 |
| | Homography Estimation | 27.1 |
| | Merge | 1 |
| | Complete | 729.0 |

**Table 6.2:** Results of speed testing

The results show that especially image loading takes a surprising amount of time. The average timings are so low because we already preload several frames at once. The actual loading step for ten consecutive frames takes up to 1067 ms / 1235 ms / 2556 ms. This has a large impact on the average duration of the complete tracking step, since every time a sequence is loaded, the rest of the tracker has to wait. This manifests in the interface in this way, that the trajectories are always build in thrusts. We believe that the he main reason for the slow image loading is that the Quicktime API is built for loading frames sequentially and continuously. As soon as we stop loading, Quicktime probably frees system resources, which have to be reacquired before the next loading step. The rest of the tracking produced very good results. The CPU is fast enough to compute SIFT features in only 25 to 113 ms. The difference is due to the number of features in the region and the size of the current search window. On the Macbook Pro we used for most of the pre-

liminary testing, computing features on the CPU takes in between 115 and 300 ms. This is where the GPU implementation would be beneficial since it works faster on the Macbook and would also remove burden from the CPU. Computation of the optical flow fields for the test videos took 1 minute and 46 seconds for a pair of frames in one direction. When employing all cores, a frame takes roughly 13 seconds. Thus, for a resolution of 960x540 the speedup over the optical flow algorithm is $\frac{13000\text{ms}}{224\text{ms}} \approx 58$. Comparison with *DimP* is not as accurate since *DimP* ran on a PC with different specifications. Computing the SIFT based flow took approximately 2.5 seconds on a Intel CoreDuo2 machine with 2 GB RAM. With this number, DRAGONEYE is $\frac{2500\text{ms}}{224\text{ms}} \approx 11$ times faster than *DimP*.

# Chapter 7

# Summary and future work

The final chapters contains an overview of our work and an outlook on what research should be done in the future.

## 7.1 Summary and contributions

This thesis presented a set of requirements for a tracking algorithm that is supposed to be used for direct manipulation. These requirements were based on experience in existing systems, as well as general technical conditions and the user. The main results were

- Invariance to object attributes
- Handling of occlusions
- Support for camera motion compensation

Based on the listed requirements, we evaluated the usefulness of several tracking techniques for our algorithm DRAGONEYE. CAMShift tracking with color histograms is useful because it can handle motion blur and form changes well. Problems can arise by the representation as a kernel,

which does not necessarily allow stable tracking of the interest point.
The SIFT point detector already proved to be useful for direct manipulation in *DimP*. This, and the high repeatability and stable positioning of feature points, even under changing form and lighting, indicated its usefulness for DRAGONEYE. This also coincides with the requirements for camera motion estimation which requires a set of points to compute a representation of the camera motion.

We used the two techniques to develop and implement DRAGONEYE, an object tracker based on point and color tracking. The algorithm is designed to handle occlusions by learning features in the vicinity that move coherently with the interest point. In this way, even full occlusions of the object of interest can be handled, if another object in the scene fits its movement because these other features can be used during the occlusion. CAMSHift tracking was implemented to overcome problems posed by motion blur and small objects. Second to tracking, we implemented homography estimation and visualization to represent camera movement in a scene, allowing for interaction that is more suitable for the user's greater understanding of the video.

Finally, we compared the quality and speed of the developed algorithm with two existing implementations employing flow fields as a basis for tracking. The results show that especially the point tracking part of the algorithm performs well. The algorithm has limits when it comes to small objects and strong non-rigid movement, impeding the development of a stable model. The algorithm is currently 50 times faster than the original DRAGON and 10 times faster than *DimP*, which corresponds to a rate of approximately 5 frames per second.

The main contributions of our work are:

- Definition of technical requirements for direct manipulation
- Development and implementation of DRAGONEYE
- Evaluation of the developed algorithm

## 7.2 Future work

Implement suitable SIFT detector

Changing object size and different levels of interest require a SIFT detector that is configurable on the fly, while being faster than the CPU implementation. Camera motion estimation and object tracking could be done with a single detection, reducing the time for this expensive step of the algorithm. Also, the image loading step has to be improved to allow for faster tracking and, even more important fast visualization of the results.

The evaluation shows that the new algorithm performs well on simple movement and partial occlusions. Improvements are necessary for videos with motion blur and rotation around the y-axis. The latter could be achieved by a gaussian weighting of point importance based on their distance and average object size. In this way, stable background points close to the edge of an object are less likely to overrule the center points that actually belong to the model. Performance during the occurrence of motion blur cannot easily be improved for the point tracking algorithm. Especially the SIFT descriptor, which is based on gradients does not stay stable in the presence of motion blur. Several scenes showed that CAMShift is a good addition to the SIFT tracking algorithm. It enables the algorithm to work, even if the object moves very slowly or SIFT features are unreliable. Unfortunately, the implementation currently lacks tracking stability. This makes DRAGONEYE very reliant on SIFT. The CAMShift implementation works with the hue component of the HSV color model, which restricts the applicable colors of objects, since grey and black do not have a unique hue. Although this works for specialized applications, direct manipulation requires more generality. For example, instead of converting the YUV color provided by the Quicktime API to HSV, CAMShift could work on the provided color components directly. This would also reduce the processing time needed for image conversion. Different techniques for detecting the original object shape, allowing for a better selection of the initial kernel could also prove useful. For example, mean-shift segmentation Comaniciu and Meer [2002] will yield regions that fit the tracker well. How well these regions represent the object that is interesting to the user has to be determined in additional tests.

Improve CAMShift implementation

Full occlusions are also still a problem. Although users will understand more easily why an algorithm fails during a full occlusion, improvements are still possible and will also increase the stability in other situations. Currently, prediction is done by assuming that the behavior of the object does not change during an occlusion. The last state the algorithm estimated is not always correct, especially when accuracy is reduced because of the partial occlusion before complete loss of the object. Thus, techniques that consider the existence of estimation errors for their prediction will produce better results. A simple stochastic filter doing this is the Kalman filter (see section 3.3.4). Another way would be to employ the user's knowledge when an object is fully occluded. As soon as the tracker detects a full occlusion, the position the user points at can be emphasized when trying to reacquire the object. It is important, however, that the user recognizes the need for pointing at the exact position, which is not necessary for the standard interaction. Also, this interaction must actually be faster than using the inertia in DRAGON to push the object over the occlusion and regrabbing it.

User tests

The most important part of future work is to test the algorithm in the wild, i.e with users and their own videos. The trajectories provided in our test videos do not, and cannot, represent any possible scenario. Furthermore, a question like "How important is the ability to track small objects for the algorithm?" cannot be answered without interface tests, and are an important subject of future work.

Camera motion estimation can be improved as well. Since we assume our image to be planar, errors will arise in some scenes. As argued before, models that consider the exact 3D representation are not viable. It is likely though, that the user is interested in only one plane at a time. Thus, in case a different plane is of interest, the user can indicate the correct plane, and the planar homography corresponding to the indicated plane can then be used for visualization.

# Appendix A

# Code Excerpts

## A.1 CoreImage Filter

This simple CoreImage filter applies a homography to an image by choosing a new sample position based on the nine parameters of an homography 4.3. samplePosition = originalPosition$^T \cdot H^T$

```
kernel vec4 applyHomography (sampler src, float m11, float ..., float m33)
{
       vec2 newCoord;
       vec2 originalCoord = destCoord ();
       newCoord.x = m11 * originalCoord.x + m21 * originalCoord.y + m31;
       newCoord.y = m12 * originalCoord.x + m22 * originalCoord.y + m32;
       float w    = m13 * originalCoord.x + m23 * originalCoord.y + m33;
       newCoord.x /= w;
       newCoord.y /= w;
       return sample (src, samplerTransform (src, newCoord));
}
```

# Appendix B

# Videos Used for Evaluation

This appendix contains images of the scenes we used for evaluation, as well as the trajectories that were not described in chapter 6.

**Figure B.1: Scene 3:** A stabilized version of scene 1 created by DRAGONEYE

Scene 4 shows two students playing beach volleyball. Both

**Figure B.2:** A stabilized version of scene 2 create by DRAGONEYE

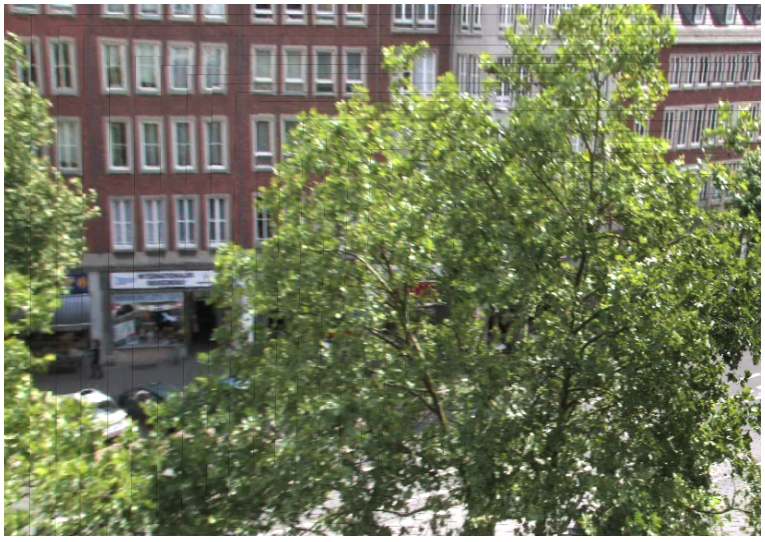

**Figure B.3: Scene 3:**

players simply move toward the net. We tested trajectories in the breast region of both players. Additionally, we tested the falling ball, which moves very fast, and is blurry.

Scene 5 shows people crossing a busy street.

**Figure B.4:** Scene 4

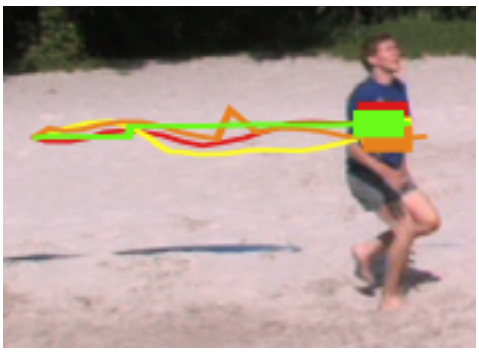

**Figure B.5: Scene 4: Player 1 trajectory** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

We tested three trajectories. The first one starting on the backpack of the man in the center.

The backpack becomes occluded by the girls head after only five frames, while the upper body and head of the man carrying it stay visible. The second trajectory starts on the couple on the left. The second interest point is never occluded and moves coherently with the couple. Both objects stay rigid over the course of the scene. A third trajectory starts on the girl's scarf. In the beginning, her movement fits that of her friend, while he slows down towards the end. Also, she turns her head away from the camera and back to it.

**Figure B.6: Scene 5:** The green dots show the starting points of the tested trajectories

**Figure B.7: Scene 5 — Couple trajectory:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

**Figure B.8:** Scene 6

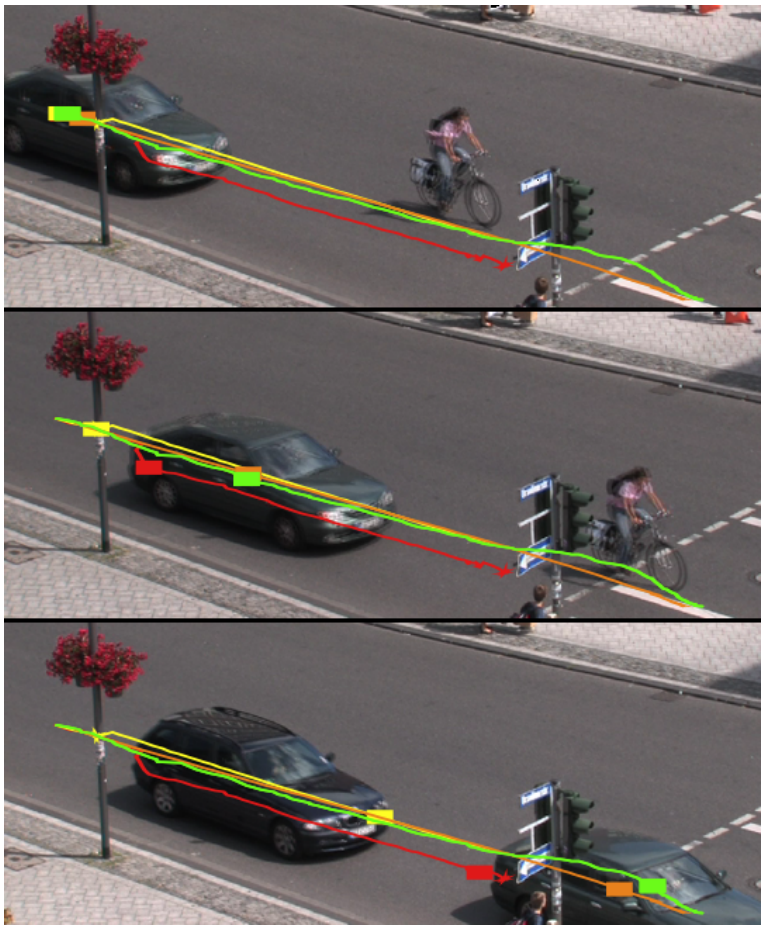

**Figure B.9: Scene 6 — First car trajectory:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

The sixth scene shows the same intersection at a wider angle. Again, three trajectories are computed; the woman on the bike, the first car, and the woman to the right of the advertising pillar moving towards the street. The car is occluded by the lights and almost stops before leaving the frame. The cyclist is almost completely occluded by the flower pot on the street light and does not show any trackable SIFT for several frames. The woman is smaller than the other objects — roughly 5% of frame width and 20% of frame height — and is also moving slowly. The latter is especially challenging for SIFT tracking, because it currently relies on movement for object detection.

**Figure B.10:** Scene 7

Scene 7 depicts a woman pushing a stroller, turning it around and back again. This kind of 3D rotation, will have a strong influence on SIFT features. Using old features will thus not be feasible. On the other hand, the difference between two frames is small and thus little features will be lost during tracking.

**Figure B.11:** Scene 8

In Scene 8, a woman moves away from camera and hence, shows very little movement. Also, a couple crosses her path from left to right for a few frames, during which she produces a partial occlusion up to their hips. We tested trajectories at breast height and hip height of the couple and the

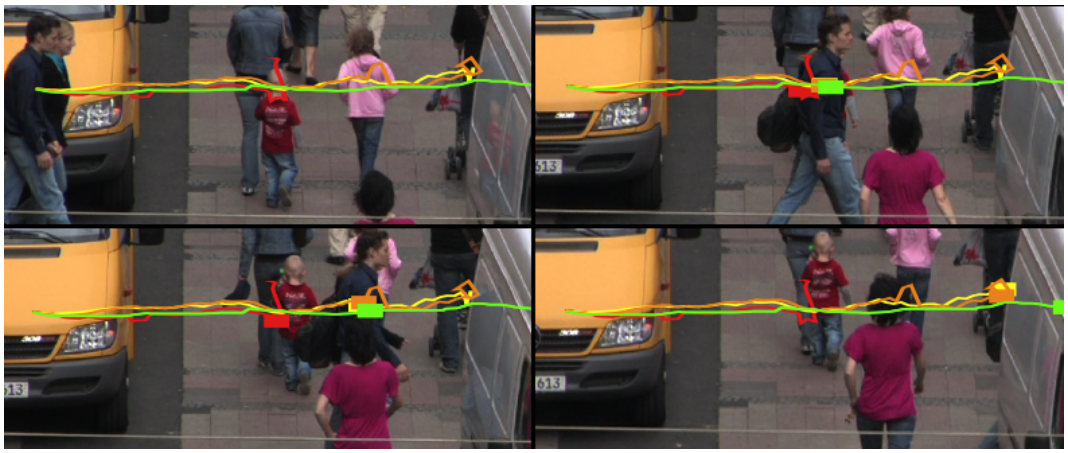

**Figure B.12: Scene 8 — Couple's breast trajectory:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

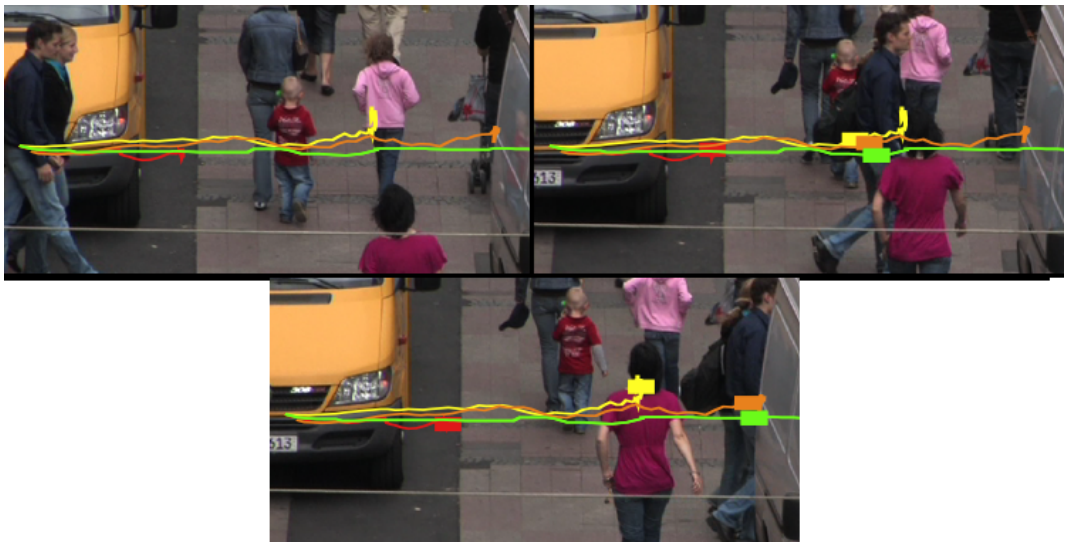

**Figure B.13: Scene 8 — Couple's hip trajectory:** Green: correct — Yellow: DRAGON — Orange: DRAGONEYE — Red: *DimP*

back of the woman.

Scene 9 depicts a basketball player doing a layup. He is followed by the camera, which produces heavy motion blur toward the end. We tested one trajectory on the players chest and a second one at the mattress leaning against the wall.

Scene 10 shows several basketball players. One trajectory

**Figure B.14:** Scene 9

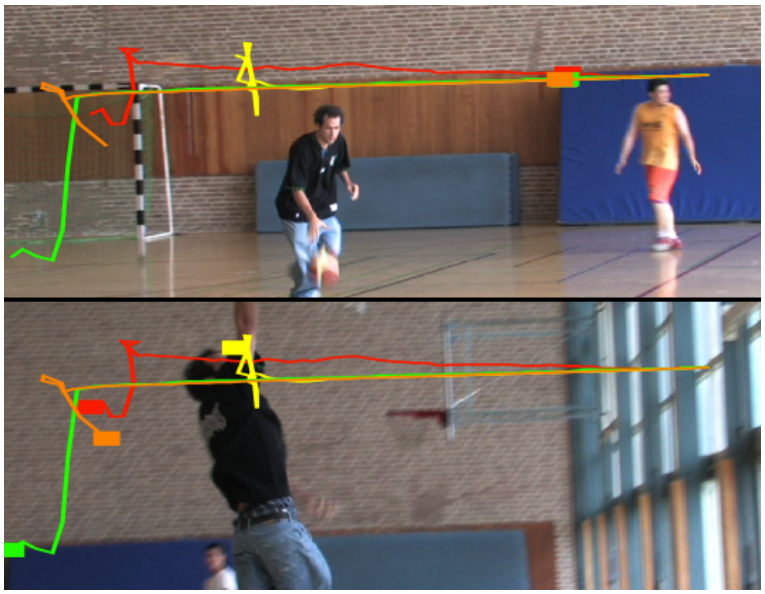

**Figure B.15: Scene 9:** Matress trajectory

on a player who is moving around the others to the basket is tested. No occlusions occur, but the scene is badly lit, the player turns several times while moving, and the camera vibrates every time the ball bounces of the ground.

In scene 11, a bus crosses the scene, taking up a large portion of the space.

**Figure B.16:** Scene 10


**Figure B.17:** Scene 11

Scene 12 shows a simple pan. We tested one trajectory on the small sign on the building saying “Babor”.


**Figure B.18:** Scene 12

**Figure B.19: Scene 12: Barbor trajectory**

# Bibliography

Opencv. URL `http://sourceforge.net/projects/opencvlibrary/`.

A. Bruhn and J. Weickert. Towards ultimate motion estimation: combining highest accuracy with real-time performance. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, volume 1, pages 749–755, 2005.

A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnorr. Variational optical flow computation in real time. *IEEE Transactions on Image Processing*, 14(5):608–615, 2005.

J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

J. Chon, T. Fuse, E. Shimizu, and R. Shibasaki. Three-dimensional image mosaicking using multiple projection planes for 3-d visualization of roadside standing buildings. *IEEE Transactions on Systems, Man, and Cybernetics*, 37(4):771–783, Aug. 2007.

O. Chum and T. Pajdla. Evaluating error of homography. In *CVWW'02: Proceedings of the Seventh Computer Vision Winter Workshop*, 2002.

D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–577, 2003.

P. Dragicevic, G. Ramos, J. Bibliowitcz, D. Nowrouzezahrai, R. Balakrishnan, and K. Singh. Video browsing by direct manipulation. In *CHI '08: Proceeding of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systemscomputing systems*, pages 237–246, New York, NY, USA, 2008. ACM.

P. Fieguth and D. Terzopoulos. Color-based tracking of heads and other mobile objects at video frame rates. In *CVPR '97: Proceedings of the Conference on Computer Vision and Pattern Recognition*, page 21, Washington, DC, USA, 1997. IEEE Computer Society.

D.B. Goldman, B. Curless, D. Salesin, and S. M. Seitz. Schematic storyboarding for video visualization and editing. In *SIGGRAPH' 06: Proceedings of the Thirty-Third International Conference on Computer Graphics and Interactive Techniques*, pages 862–871. ACM, 2006.

C. Harris and M. Stephens. A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988.

Richard Hartley and Andrew Zisserman. *Multiple view geometry*. Cambridge University Press, New Yorck, second edition, 2000.

B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 16(1–3):185–203, August 1981.

C. Hsu, T. Cheng, R.A. Beuker, and J. Horng. Feature-based video mosaic. In *ICIP '00: Proceedings of the International Conference on Image Processing*, volume 2, pages 887–890, 2000.

D.P. Huttenlocher, J.J. Noh, and W.J. Rucklidge. Tracking non-rigid objects in complex scenes. In *ICCV '93: Proceedings of the Fourth International Conference on Computer Vision*, pages 93–101, 1993.

S. Ince and J. Konrad. Occlusion-aware optical flow estimation. *IEEE Transactions on Image Processing*, 2008.

M. Irani, P. Anandan, and S. Hsu. Mosaic based representations of video sequences and their applications. In *ICCV '95: Proceedings of the Fifth International Conference on Computer Vision*, pages 605–611, 1995.

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

D. Kimber, T. Dunnigan, A. Girgensohn, F. Shipman, T. Turner, and Tao Yang. Trailblazing: video playback control by direct object manipulation. In *ICME '07: Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 1015–1018, 2007.

S. Lazebnik, C. Schmid, and J. Ponce. A sparse texture representation using affine-invariant regions. In *CVPR '03: Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2003.

B. Li, R. Chellappa, Q. Zheng, and S.Z. Der. Model-based temporal object verification using video. *IEEE Transactions on Image Processing*, 10(6):897–908, 2001.

T. Lindeberg. *Discrete scale-space theory and the scale-space primal sketch*. PhD thesis, 1991.

T. Lindeberg. Scale space theory: a basic tool for analyzing structures at different scale. *Journal of Applied Statistics*, 1994.

D.G. Lowe. Object recognition from local scale-invariant features. In *ICCV '99: Proceedings of the Seventh International Conference on Computer Vision*, volume 2, page 1150. IEEE Computer Society, 1999.

D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2): 91–110, 2004.

K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *ICCV'01: Proceedings of the Eights IEEE International Conference on Computer Vision*, volume 1, pages 525–531, 2001.

K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.

J. Raskin. *The humane interface*. ACM Press, 2000.

S. Schuon and K. Diepold. Comparison of motion deblur algorithms and real world deployment. In *57th International Astronautical Congress*, 2006.

D. Serby, E.K. Meier, and L. Van Gool. Probabilistic object tracking using multiple features. In *ICPR '04: Proceedings of the Seventeenth International Conference on Pattern Recognition*, volume 2, pages 184–187, 2004.

S.N. Sinha, J. Frahm, M. Pollefeys, and Y. Genc. Gpu-based video feature tracking and matching. Technical report, Department of Computer Science, UNC Chapel Hill, 2006.

F. Tang and H. Tao. Object tracking with dynamic feature graph. In *PETS '05: IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, 2005.

M. Unger. Object detection in video sequences. Master's thesis, RWTH Aachen University, 2004.

A. Vedaldi. siftpp. URL `http://web.me.com/vedaldi/code/siftpp/siftpp.html`.

M. Weiss. Depth-discontinuity preserving optical flow using time-multiplexed illumination. Master's thesis, RWTH Aachen University, University of Southern California, 2007.

G. Welch and G. Bishop. An introduction to the kalman filter. In *SIGGRAPH' 01: Proceedings of International Conference on Computer Graphics and Interactive Techniques*, 2001.

A. P. Witkin. Scale-space filtering. In *IJCAI '83: Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1019–1022, 1983.

C. Wu. Siftgpu. URL `http://cs.unc.edu/~ccwu/siftgpu/`.

A. Yilmaz, Xin Li, and M. Shah. Contour-based object tracking with occlusion handling in video acquired using mobile cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1531–1536, 2004.

A. Yilmaz, O. Javed, and M. Shah. Object tracking: a survey. *ACM Computing Surveys*, 38(4):13, 2006.

# Index

Typeset October 30, 2008