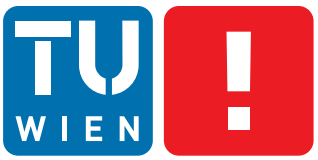


FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Object Tracking using Projective Invariants

Clemens Korner

Computer Vision Lab
Institute of Computer Aided Automation
Vienna University of Technology

January 31, 2016

Supervisors: Robert Sablatnig
Roman Pflugfelder

Abstract

Tracking of objects without a priori knowledge of the object class is a growing topic in computer vision. One popular approach is to use multiple interest points for describing the position of the object. An essential step for these methods is the estimation of the position of the points in a new image. Optical flow is a method to calculate the displacements of these points, however it does not produce in all cases satisfactory predictions.

In this thesis we show a new method to detect these outliers on planar objects. For that we take advantage of projective invariant properties of these points. With the properties the algorithm is able to recognize wrong position estimations and can consider them in the further tracking process. The verification of the projective invariant values can be efficiently done and therefore the method processes the images with more than 160 fps. By comparing with eight state-of-the-art trackers, we show that our approach has the best trade-off between computation speed and tracking precision on a standard PC. For our experiments we use 25 sequences of a well established benchmark dataset to compare our method with other trackers and show its strengths and improvement possibilities.

1 Introduction

In the 1960s *Lawrence G. Roberts* analyzed the extraction of 3D information from 2D views on a world consisting of blocks. He published his results in his Ph.D. thesis [Roberts, 1963] at the the Massachusetts Institute of Technology [Huang, 1996]. Gilbert Falk responded to his publication and improved his proposed methods [Falk, 1972]. This was also the start of *computer vision*, the attempt to write algorithms which make images and their contents understandable for machines [Bennamoun and Mamic, 2012].

Object tracking is one subfield of computer vision. The goal thereby is to write algorithms which are able to track objects in videos and calculate their location in each image frame [Maggio and Cavallaro, 2011]. Due to unwanted real world effects like occlusions, abrupt motion, pose and viewpoint changes, noise and blur, object tracking is a challenging field [Yilmaz et al., 2006]. The origins of object tracking are found in *radar tracking*. The response of a radar signal is superimposed by unwanted radar returns from other objects like trees and buildings. The problem in this field is to find the response from the target of interest [Ramachandra, 2000]. The pioneer of radar tracking is Robert W Sittler [Ramachandra, 2000] with his article "An optimal data association problem in surveillance theory" [Sittler, 1964].

1.1 Motivation

In the 40s of the 20th century James J. Gibson, an American psychologist, delved deeply to describe the way animals perceive visual stimulus [Hochberg, 1994]. Some years later in 1950 he published his results in the book *The Perception of the Visual World* and introduced a new concept called *optical flow* [Gibson, 1950]. Bats and birds such as pigeons and goshawks use optical flow to navigate their flights through forest cutter [Sebesta and Baillieul, 2012]. With the publication of Lucas and Kanade in 1981 [Lucas and Kanade, 1981] this technique was also introduced in the computer vision area and is used in trackers [Nebehay and Pflugfelder, 2014, Kalal et al., 2010, Vojir and Matas, 2014]. But this algorithm is not able to predict the exact motion of every point in an image [Lucas and Kanade, 1981]. We implement a new tracking method, which calculates *projective invariant* values for these points in motion. With the help of these invariant values we remove wrong correspondences.

1.2 Contributions

We use the proposed method *Good Features to Track* [Shi and Tomasi, 1994] to find interest points in the initial image. For all points we calculate projective invariant values. Mundy et al. explain the usage and properties of these values theoretically in their work [Mundy and Zisserman, 1992]. Our algorithm estimates the displacement of the key points between two sequential images by *Lucas and Kanade Sparse Optical Flow* [Lucas and Kanade, 1981]. After this step we determine projective invariants again to find key points which have been tracked wrongly. These outliers will be removed from the key point database. For the estimation of the new state of the object we use *RANSAC* [Fischler and Bolles, 1981].

We implemented our idea in *Python* and evaluate it with a dataset containing several scenes and compare the method with other tracking methods.

1.3 Problem Definition

The problem of object tracking can be best defined in the words of *Alper Yilmaz* "In its simplest form, tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene." [Yilmaz et al., 2006]

In all images $I_1 \dots I_n$ of a video sequence a tracker calculates an estimation of the position of a target. This position is called state and can be captured as vector

$$\mathbf{x}_i = \begin{pmatrix} tl_x \\ tl_y \\ br_x \\ br_y \end{pmatrix} \tag{1.1}$$

that represents for example a rectangle. These four coordinates describe the rectangle that enclosures the target. tl_x and tl_y represent the x any y coordinate of the top left corner of the rectangle, br_x and br_y represent the coordinates of the bottom right corner. It is assumed that the rectangle has no rotation and its sides are parallel with the x respectively y axis. Therefore it is possible to describe this quadrilateral with only two coordinates. The index i of the state $\mathbf{x}_i$ stands for the corresponding frame number of the state. $\mathbf{x}_1$ is the state of the target in the first frame, $\mathbf{x}_n$ the state in the last frame.

$$i \in \{1, 2, \dots, n-1, n\} \tag{1.2}$$

For the calculation of the next state we presume the *Markov assumption*. A method that satisfies the Markov assumption only needs the last state $\mathbf{x}_k$ to calculate the next state $\mathbf{x}_{k+1}$. The previous states $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{k-1}$ are not required [Gardiner et al., 1985].

Our focus in this work is on *single-target tracking*. In this case there is only one target tracked and we can optimize our approach in this simplified scenario. With these experiences the tracker can be extended for multiple targets. One possibility to define the object to be tracked and how the initialization of the algorithm is done, is to provide a list of keywords. Another alternative is to initialize the tracker with the bounding box of the target in the first frame. Our tracker will need the state $\mathbf{x}_1$ of the target in the first frame during the initialization [Forsyth and Ponce, 2002, Yilmaz et al., 2006].

Summarized the problem can be seen in Figure 1.1. The current state $\mathbf{x}_k$ of one single target is given. The task is to calculate the state in the next frame $\mathbf{x}_{k+1}$.

Object Tracking Algorithm

In Figure 1.2 the basic concept of a single-target tracker is visualized. At the beginning the tracker needs to be *initialized*. Therefore additional information, like the first state or parameters of the sequence are required. With this data and the first image, the tracker calculates a representation of the target. This representation is crucial for the whole tracking task, because the algorithm must distinguish the target from different objects in the scene.

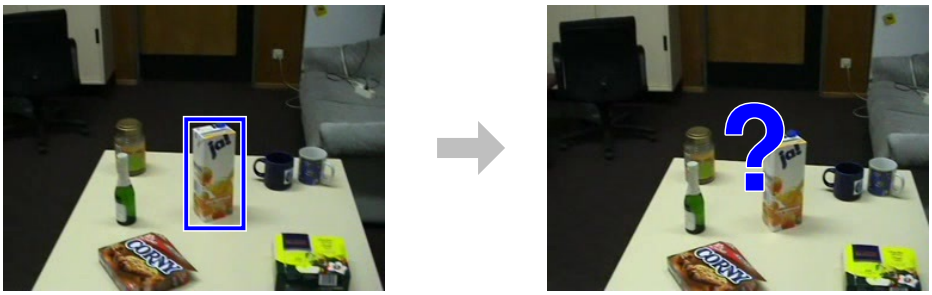


Figure 1.1: Object Tracking; the blue rectangle on the left represents the state $\mathbf{x_k}$, the task is to find the state $\mathbf{x_{k+1}}$ in the following image

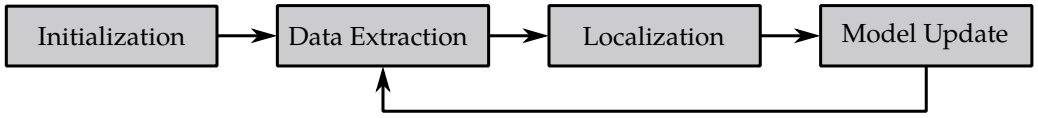


Figure 1.2: Object Tracking Algorithm

With the help of the model, the tracker is able to estimate all states for the following frames. For every image of the sequence the tracker executes the remaining three steps sequentially. First the algorithm *extracts data* of the new image and prepares them for further usage. The result of this extraction can be e.g. color histograms, the displacement of the pixels, et cetera.

The tracker combines all the extracted data with the previous state. With this combination the method predicts the *localization* of the target and thereby the new state.

The algorithms are able to join the information of the state and the new estimated state to *update the model*. Herby the representation captures new information of the target, which were not present during the initialization. But extensive updates must be treated with caution, because they can falsify the model and thereby all the following state estimations [Bradski and Kaehler, 2008, Challa, 2011].

1.4 Thesis Structure

This thesis is divided into four chapters. These chapters with their names and a short explanation are listed below.

Chapter 2: *Related Work* provides information about existing approaches and explains advantages and disadvantages of current state of the art methods. Additionally it clarifies the properties and origins of *projective invariants* and their mathematical description is discussed.

Chapter 3: *Methodology* explains the implementation of our algorithm in more detail and illustrates its functionality.

Chapter 4: *Results* describes the used dataset, shows the results of our algorithm and compares them with other methods.

Chapter 5: *Conclusion* summarizes the work and discusses possible future work.

1.5 Summary

In this chapter we gave an overview about computer vision and object tracking. After that we described the motivation behind this thesis and explained the concept of optical flow and its usages in object tracking. Then we gave a problem definition and illustrated the concept of single-target tracking. We described the structure of this thesis and explained the chapters of this work.

2 Related Work

Chapter one gave a first insight into the history of computer vision and basic concepts. It would fill several books to explain all concepts of this field in full detail. So this chapter focuses on the concepts and articles of object tracking which influenced our work. If the reader is interested in more computer vision fields, we would suggest [Hartley and Zisserman, 2003, Bradski and Kaehler, 2008]. They are a good point to start.

This chapter has two additional parts at the end, Section 2.2 and Section 2.3. They summarize the necessary theory of these concepts, which are needed to understand the rest of the thesis.

2.1 Projective Invariants and Object Tracking

The first mentionable article about projective invariants in computer vision was published in the year of 1992 by Mundy et al. [Mundy and Zisserman, 1992]. It explains the theoretical concepts of projective geometry and their invariants. Section 2.2 summarizes the crucial concepts of this work, which are important for our algorithm.

We are not aware of any article which is about object tracking by exploiting projective invariants. However there are two papers, which use projective invariants for a specialized tracking application, *marker tracking*. In virtual and augmented reality systems markers are used to calculate the pose between a camera and an object [Kohler et al., 2011]. In [Loaiza et al., 2007] the authors use the projective invariant values to distinguish dissimilar marker patterns. They have four collinear and five coplanar patterns in their configuration. The other work [van Liere and Mulder, 2003] shows how markers with projective invariant patterns can be used to reduce the search space for the recognition of the markers images.

The next enumerations describe algorithms which can be used for tracking. A algorithm for point tracking is optical flow [Lucas and Kanade, 1981]. This method is able to estimate the sparse optical flow of an image patch and thereby the displacement of the patch. We use this algorithm for the motion calculation of interest points between two sequential images. *Mean shift* is another tracking approach. In 2000 Comaniciu et al. published a paper [Comaniciu et al., 2000], which explains how to develop a tracker with the aid of mean shift. First of all they calculate the color histogram of the target. Afterwards they make an iterative maximization of a similarity function in each image and estimate the new state.

The combination of interest point detectors and descriptors can be used for object trackers [Nebehay and Pflugfelder, 2014]. An interest point detector finds distinctive points, e.g. *Good Features to Track* [Shi and Tomasi, 1994] or *Harris Corner Detector* [Harris and Stephens, 1988]. Tuytelaars et al. proposed an article [Tuytelaars and Mikolajczyk, 2008] about strengths and weaknesses, the history and the evolution of feature detectors. These interest points and their surrounding pixels are characterized by descriptors, for instance BRISK [Leutenegger et al., 2011], ORB [Rublee et al., 2011] SURF [Bay et al., 2008] and SIFT [Lowe, 2004]. The characterization vectors are called *feature vectors*. In the next image this process is repeated again. A matcher finds the corresponding feature vectors of the two images. With these correspondences it

is possible to calculate the new state. Due to the fact that the matching process is not perfect, there is the possibility that some matches are wrong. Random sample consensus (*RANSAC*) [Fischler and Bolles, 1981] is a non deterministic algorithm to detect outliers. This method uses only a small part of the matched key points to calculate a fitting transformation of the state. Afterwards this transformation is compared to the movement of the remaining point combinations. This process separates the interest points into inliers and outliers. These steps are repeated a fixed number of times. The transformation with the biggest number of inliers is applied to the state.

The CMT tracker [Nebehay and Pflugfelder, 2015] enhances these detection methods with a recursive tracking approach and an additional clustering of correspondences. In comparison TLD [Kalal et al., 2012] uses image patches instead of interest points. The tracker contains also a learning concept, where it tries to identify detection errors and updates them to avoid these errors in following images. They call their learning method P-N learning. Kalal et al. applied a forward- backward-measurement on the tracked points. This measurement reveals erroneous tracked interest points. During the initialization, all points are arranged in a rectangular grid. Vojir et al. extend this method in [Vojir and Matas, 2014]. Each point may move in a restricted area. Only if the key point leaves this area it will be reinitialized.

2.2 Projective Invariants in Detail

In this section we explain theories of projective geometry and projective invariants.

If someone is making a picture, the coordinates are transformed from real world 3D-coordinates to 2D-coordinates in the picture. This procedure is known as *3D to 2D camera projection*. A different necessity is to calculate the transformation rule, to transform the coordinates from an object in one image to its coordinates in another image, also called *2D homography* [Hartley and Zisserman, 2003].

The transformation must be able to calculate the coordinates in an image of an infinite distant point, e.g. the horizon. Euclidean geometry is not powerful enough to do this transformation, therefore a more powerful mathematical theory is necessary. *Projective Geometry* with the corresponding *Projective Transformations* is able to handle these cases. An example of a projective transformation can be seen in Figure 2.1. But these transformations do also have disadvantages. Parallelism and orthogonality are not necessarily present anymore and the distance between two points is also affected by the coordinate transformation. Parallel lines do not stay parallel, they meet each other in a common point, the vanishing point. Some properties however are not affected by projective transformations which are called *Projective Invariants* [Mundy and Zisserman, 1992].

2.2.1 Fundamentals and Hierarchy

A projective transformation can be described by a matrix multiplication of the transformation matrix $\mathbf{H}_p$ by a vector $\mathbf{x}$ holding the coordinates to be transformed.

$$\mathbf{x}' = \mathbf{H}_p \mathbf{x} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v}^{\mathrm{T}} & v \end{bmatrix} \mathbf{x} \tag{2.1}$$

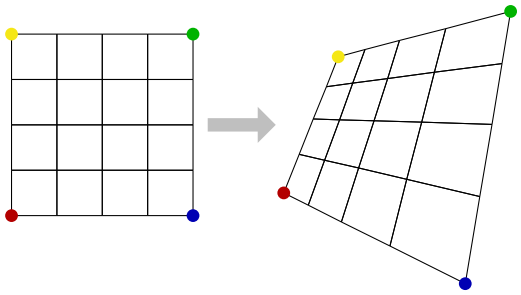


Figure 2.1: An example of a projective transformation

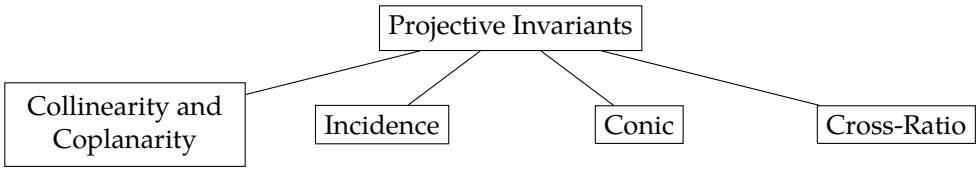


Figure 2.2: Hierachy of Projective Invariants

The transformation matrix has nine different entries. Only the ratio of the eight elements is required, therefore the ninth element v is mostly scaled to unity. $\mathbf{t}$ is a translational 2-vector, $\mathbf{A}$ is a 2×2 non-singular matrix to calculate the rotation and deformation of the object and $\mathbf{v}$ is a 2-vector used to describe the non-linear effects of the projective transformation. For these transformations *homogenous coordinates* are used. With homogenous coordinates the transformations become linear.

Several different projective relations exist which are invariant to projective transformations. The hierarchy of these are shown in Figure 2.2. The following sections describe these preserved properties. The assumption for all the specified invariants in this chapter is, that the points and lines are always located on planar surfaces [Mundy and Zisserman, 1992, Hartley and Zisserman, 2003].

The next sections define those projective invariants which are relevant for our approach. In [Mundy and Zisserman, 1992] is an elaborate description of conics and their behavior related to projective transformations.

2.2.2 Collinearity and Coplanarity

A set of points is called collinear, if all points lie on the same line. Collinearity is not only limited to Euclidean- and projective geometry, it is also a concept in spherical geometry. But the meaning is different, the points are not located on a line in the classical sense, they are arranged on circles [Belot, 2011]. Two points in $\mathbb{R}^2$ are always collinear.

According to Bronstein et al. [Bronstein and Semendjajew, 1979] the collinearity of three points in the two dimensional space

$$\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbb{R}^2$$

can be checked as follows

$$\begin{vmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \\ 1 & 1 & 1 \end{vmatrix} = 0 \tag{2.2}$$

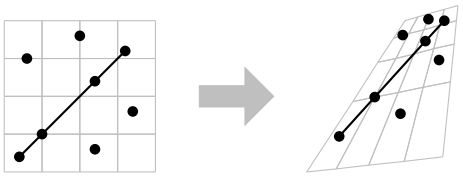


Figure 2.3: Projective invariance of collinear and coplanar points

If the determinant is not equal to zero the points are not collinear. A set of n points in $\mathbb{R}^m$ are collinear if the point-line distances of the points $\mathbf{v}_3, \ldots, \mathbf{v}_n$ to the line defined by $(\mathbf{v}_1, \mathbf{v}_2)$ are for all points $3, \ldots, n$ zero.

Several points are coplanar if a (hyper) plane exists so that all points lie on this plane. Three points in $\mathbb{R}^3$ are always coplanar.

As explained by Bronstein et al. [Bronstein and Semendjajew, 1979] Equation (2.2) is a simplification of a more general equation. The generalization is used to check if four points in the three dimensional space

$$\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4 \in \mathbb{R}^3$$

lie on the same plane.

$$\begin{vmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \mathbf{w}_3 & \mathbf{w}_4 \\ 1 & 1 & 1 & 1 \end{vmatrix} = 0 \tag{2.3}$$

A number of n points in $\mathbb{R}^m$ are coplanar if the point-plane distances of the points $\mathbf{w}_4, \ldots, \mathbf{w}_n$ to the plane defined by $(\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$ are for all points $4, \ldots, n$ zero. Figure 2.3 visualizes the consequences of collinearity and coplanarity. After a projective transformation collinear points (on the black line) remain collinear and coplanar points (in the gray square; including the collinear points) remain also coplanar.

2.2.3 Incidence

The incidence criteria checks if three or more lines are concurrent. Lines are concurrent if they intersect each other in the same point. Two non parallel lines have always one intersection point. Figure 2.4 shows the invariance of an incidence to projective transformations.

Three lines, described with line equations

$$\begin{aligned} a_1 x + b_1 y + c_1 &= 0 \\ a_2 x + b_2 y + c_2 &= 0 \\ a_3 x + b_3 y + c_3 &= 0 \end{aligned} \tag{2.4}$$

are concurrent if their determinant satisfies

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = 0 \tag{2.5}$$

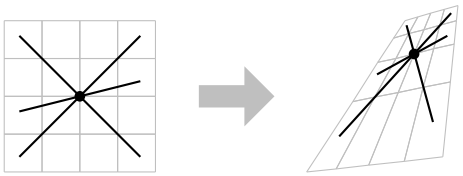


Figure 2.4: Projective invariance of the incidence

A set of n lines in $\mathbb{R}^m$, each defined by two points $(\mathbf{v}_{11}, \mathbf{v}_{12}), \ldots, (\mathbf{v}_{n1}, \mathbf{v}_{n2})$, are concurrent, if all the combinations of line one with every other line have the same intersection point. The lines are determined in parametric form.

$$\mathbf{l}_n = \mathbf{v}_{n1} + (\mathbf{v}_{n2} - \mathbf{v}_{n1}) \cdot t_n \tag{2.6}$$

The parameters $t_1, \ldots, t_n$ can be obtained by inserting line one in the other $n - 1$ equations. If there is always a consistent solution for t_1 and the value is always the same, all lines are concurrent [Heuel, 2004].

2.2.4 Cross-Ratio

There are several different cross-ratios defined, which are included in projective invariants. A cross-ratio is the product of two ratios, therefore each cross-ratio needs exactly four distinct values e.g. coordinates of points, angles between lines or distances between points. All the following cross-ratios are invariant to projective transformations [Mundy and Zisserman, 1992].

Collinear Points

The cross-ratio of the distances of four collinear points is defined by the equation

$$\mathrm{CR}(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}) = \frac{\overline{AC}}{\overline{BC}} \cdot \frac{\overline{BD}}{\overline{AD}} \tag{2.7}$$

where $\overline{XY}$ presents the Euclidean distance between point **X** and point **Y**. Figure 2.5a shows the order of the points [Mundy and Zisserman, 1992].

Concurrent Lines

The angle of four concurrent lines can be used to characterize a cross-ratio

$$\mathrm{CR}(A, B, C, D) = \frac{\sin(\angle AC)}{\sin(\angle BC)} \cdot \frac{\sin(\angle BD)}{\sin(\angle AD)} \tag{2.8}$$

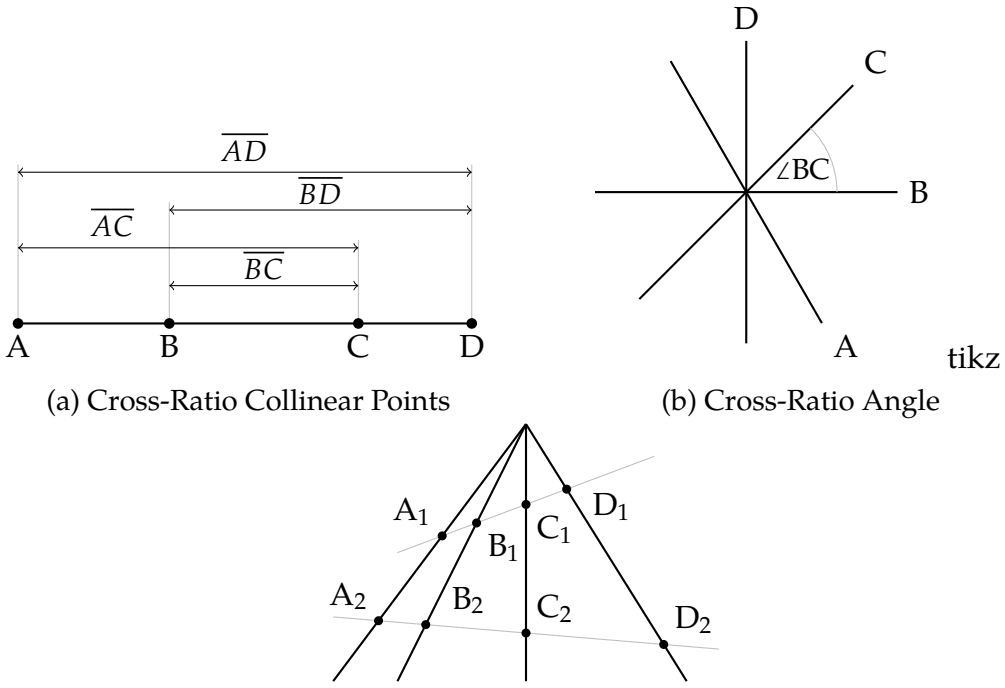
$\angle AC$ is the smallest angle between line A and line B. See Figure 2.5b for more details [Mundy and Zisserman, 1992].

Collinear Points on Concurrent Lines

For all lines cutting the pencil of four lines, the cross-ratio remains the same. Figure 2.5c shows the exact arrangement. With Equation (2.7) the phenomenon is described by

$$\mathrm{CR}(\mathbf{A}_1, \mathbf{B}_1, \mathbf{C}_1, \mathbf{D}_1) = \mathrm{CR}(\mathbf{A}_2, \mathbf{B}_2, \mathbf{C}_2, \mathbf{D}_2) \tag{2.9}$$

[Mundy and Zisserman, 1992]



(a) Cross-Ratio Collinear Points

(b) Cross-Ratio Angle

(c) Cross-Ratio Collinear Points on Concurrent Lines

Figure 2.5: Cross-Ratios

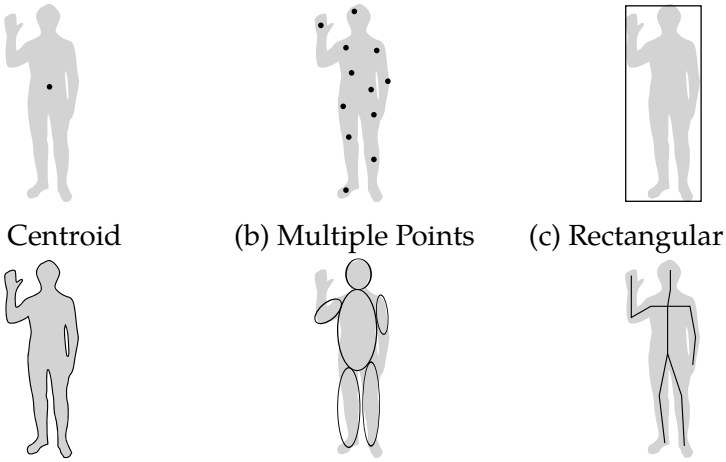
2.3 Object Tracking in Detail

This sections explains two important concepts of object tracking, the *object state* and *feature selection*.

2.3.1 Object State

Chapter 1 described rectangles for state representations. The rectangle is the model and a concrete instantiation of the rectangle with values is the representation. The number of possible instantiations defines the different possible states of the model. The more the higher is the amount of information [Shannon and Weaver, 1949]. But there are also other possibilities [Yilmaz et al., 2006].

- **Points**: One point, the centroid (Figure 2.6a), or many distributed points (Figure 2.6b) are used to characterize an object.
- **Primitive geometric shapes**: For rigid objects circles, rectangles (Figure 2.6c), ellipses are suitable representations.
- **Object silhouette**: The silhouette (Figure 2.6d) of an object is represented by its outer contour.
- **Shape and skeletal models**: Humans are assembled by parts with arms, legs, torso and the head. An appropriate model could be the composition of primitive geometric shapes (Figure 2.6e) or its skeletal (Figure 2.6f) structure.



(a) Centroid (b) Multiple Points (c) Rectangular Patch

(d) Object Contour (e) Multiple Part Patches (f) Object Skeleton

Figure 2.6: Object Representations (Human Image from [Sagan, 2013])

2.3.2 Feature Selection

Different objects have to be distinguished from each other, therefore features as unique as possible have to be extracted. These features are also part of the model [Yilmaz et al., 2006].

Color and its histogram are used to generate features for objects. Different color spaces like *RGB* (red, green and blue), *HSV* (hue, saturation and value) and *CIELUV* exist. *Edges* of objects are less sensitive to illumination changes than the color distributions and are therefore an alternative possibility [Yilmaz et al., 2006]. *Textures*, a set of texture elements occurring in some repeating pattern, are also less illumination variant features than color values [Yilmaz et al., 2006].

2.4 Summary

In this chapter we explained the related work of this thesis and declared the concepts and articles of object tracking which influenced our work. We then described theories of projective geometry and projective invariants. Thereby we explained the terms collinearity, coplanarity, incidence, and three cross-ratios. After that we gave an insight into the concepts of object tracking the and the feature selection.

3 Methodology

In this chapter we describe the details of our approach. All of our implementations follow roughly the same characteristics. Therefore the first section gives an overview about their procedure. The following sections characterize the specifics of our attempts in chronological order. Our algorithm uses concepts like the *ray casting method* and *Good Features to Track*. They are described later in the appendix of the thesis.

3.1 Overview

Based on the tracking diagram from the first chapter, we explain our approach step by step. An outlier following the scheme of Figure 1.2 is shown in Figure 3.1.

For the *initialization* process the tracker needs the state of the object of interest in the first frame $\mathbf{x}_1$. The image and the state are used to compute interest points in the first frame. We use *Good Features to Track* [Shi and Tomasi, 1994] for this task, because it detects points which have large gradients in two orthogonal directions. Since *Lucas and Kanade Sparse Optical Flow* determines the flow in the gradient direction, the gradients of the points must be large enough to be over the noise level of the image [Eltoukhy and Salama, 2002, Lucas and Kanade, 1981]. During the next step, the method chooses combinations of these points which have *projective invariant* properties. Those identified properties are stored in arrays. There is also the option that the algorithm calculates feature vectors for each key point. This is the case, when the feature to find lost key points again is enabled. More on this topic later. The collected data is used to define a model of the target.

After this initial processing, the tracker estimates the position of the object in every frame. This is done with the help of the model. Next the new image is analyzed and data for the optical flow is extracted. We use *Lucas and Kanade Sparse Optical Flow* [Lucas and Kanade, 1981] to estimate the displacement of each interest point, because it produces accurate results [Baker et al., 2011, Thota et al., 2013] and implementations for the GPU (Graphics Processing Unit) exist [Mahmoudi et al., 2014]. This algorithm blurs the two steps *data extraction* and *localization*. Optical flow uses the previous and the new image and the prior interest point coordinates to localize all these points in the new image. So optical flow both extracts the data and localizes the points.

With the new positions of the points the method examines the prior calculated projective invariant properties on the new constellation of interest points. If points do not fulfill these properties anymore, they are removed from the tracking process. An additional option to find former removed points exist. This idea is discussed in a later section. The algorithm represents the object of interest with two different states. The first and primary state is defined by the coordinates of the key points. The second one, a bounding box which enclosures the target, can be calculated with the values of the initial state $\mathbf{x}_1$ (a rectangle), the initial position of the interest points and the new positions. The bounding box is necessary for the comparison of our algorithm with the other ones, because rectangles are a de facto standard for the target state [Wu et al., 2013]. We use RANSAC [Fischler and Bolles, 1981] for the calculation of a quadrilateral and determine the smallest rectangle which enclosures

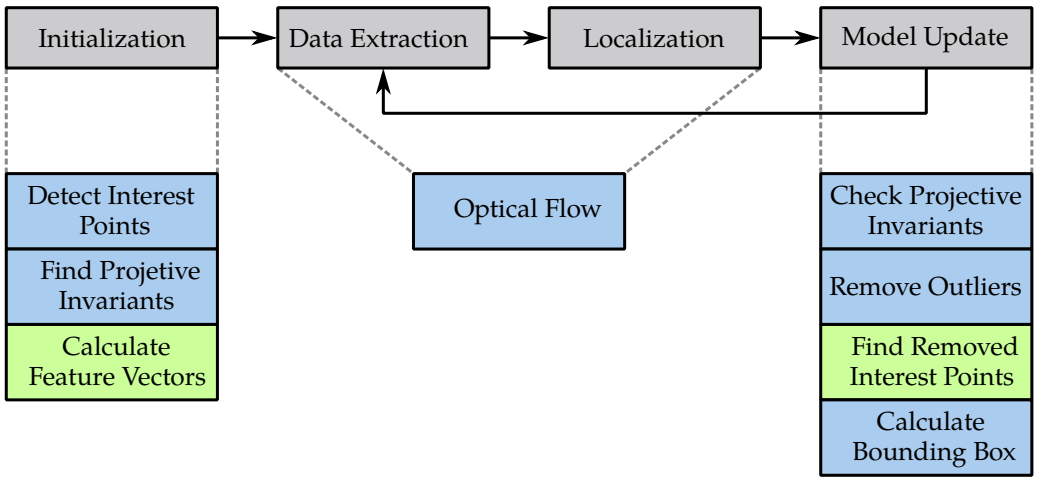


Figure 3.1: Our Algorithm; the green steps are optional, the blue ones are necessary

it. With the position of four points in the first image and their position in the new one we compute the transformation matrix (see Equation (2.1)). With this matrix the original bounding box is then transformed. RANSAC picks those four points, which are representative for the movement of all points. The model is updated with the new position of the interest points. The outliers are marked so they will not be considered in the next images anymore. The method repeats these steps for all following images.

3.2 Random Connections

Starting with this section we describe the different implementations. The structures of the algorithms are roughly the same and have been explained in the section before.

In our first attempt we calculated lines in the *initialization* stage by connecting randomly chosen pairs of interest points. The calculation is done efficiently by *Bresenham's line algorithm*. The algorithm calculates the coordinates of the lines in a discrete coordinate system [Nievergelt and Hinrichs, 1993]. Following the Python-code for Bresenham's line algorithm for the first, fourth, fifth and eight octant.

Listing 1: Bresenham's Line Algorithm

```
1 def bresenham_line(p1, p2):
2   # first point coordinate: x
3   # second point coordinate: y
4   dx = abs(p2[0] - p1[0])
5   dy = abs(p2[1] - p1[1])
6   # signum of dx and dy
7   sx = np.sign(p2[0] - p1[0])
8   sy = np.sign(p2[1] - p1[1])
9   d = 2 * dy - dx
```

```
10    x = p1[0]
11    y = p1[1]
12    points = [(x, y)]
13    while x != p2[0]:
14    if (d > 0) or ((d == 0) and (sy == 1)):
15      y += sy
16      d -= 2 * dx
17      x += sx
18      d += 2 * dy
19      points.append((x, y))
20    return points
```

For the other four octants the calculation is also possible. But the x and y coordinates of the input points, and afterwards the coordinates of the output points, have to be swapped.

The scoring function of *Good Features to Track* [Shi and Tomasi, 1994] evaluates these computed points. See Appendix A for more information about the scoring function. All but the two best scored points and the initial points of the line are removed. Figure 3.2a displays an example of the procedure. The light- and dark gray filled pixels represent together the digitalized points of the line. The endpoints of the line are striped. Under consideration of the corner detector and its scoring, only the dark gray pixels endure.

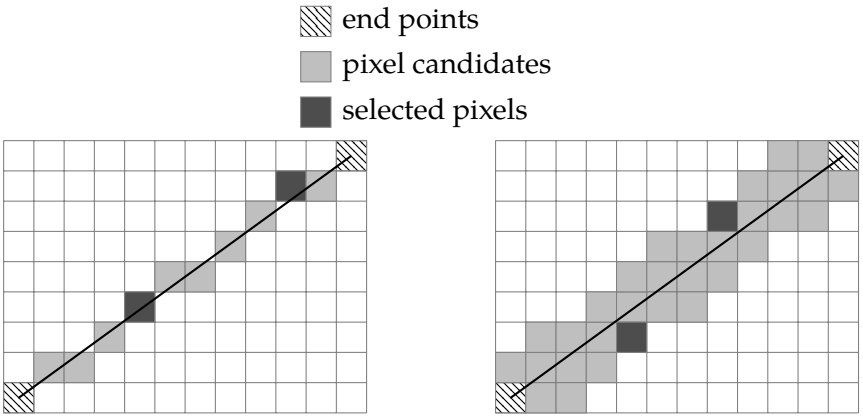
After the estimation of the displacement of each point we check all four points of each line, if they are still almost collinear. Almost collinear points are allowed to be located within a threshold around a line. We call these almost collinear points "pseudo-collinear". This term is more accurately discussed in the following section. To check for the pseudo-collinearity, we use all six combinations of the four points to create a line and measure the distances from all points to this line. If at least one combination exists, where all points are in the threshold, the key points remain active. Otherwise all four points are marked inactive and are not used anymore in the following frames for tracking.

Experiments revealed that the two best scored points located between their initial endpoints are often not reliable and yield to poor motion estimations by optical flow. On the line is only a very limited number of points and often, even if corners are near, the Bresenham's line does not include these points. Another issue is that although points will be noticed when drifting away normal to the line, shifts in the direction of the line are not recognized.

3.3 Pseudo-Collinearity

Before we begin with the description of the next algorithm, we insert an explanation of *pseudo-collinearity* in more detail. This terminology is similar to the expression "almost collinear". Collinearity is one of several projective invariants. Their description and illustration can be found on page 7 in Section 2.2.2.

In order for a number of points to be collinear, they must be *exactly* located on the same straight line. Practically through digitalized coordinates, noise and small shifting errors, previous collinear points tend to lose this property. Therefore the classic collinearity criteria is too rigorous. For that reason we introduce the



(a) Bresenham's Line (b) Bresenham's Line Additional Pixels

Figure 3.2: The figures illustrate possible points on the Bresenham's line on the left and points nearby the line on the right

terminology *pseudo-collinearity*. Pseudo-collinear points do not have to be positioned on the same line anymore, however they have to stay in an area with a defined size. A maximum threshold d exists between the desired line and the points, to tell inliers and outliers apart. All points $p_1 \dots p_k$ with their Euclidean distance $d_1 \dots d_k$ smaller than d belong to the pseudo-collinear area, all other points $p_{k+1} \dots p_n$ not. Mathematically all points

$$P = \{p_1, p_2, \dots, p_n\} \tag{3.1}$$

$$PC(ax + by + c = 0, d) = \{\forall p \in P \mid \text{distance}(ax + by + c = 0, p) \leq d\} \tag{3.2}$$

belong to the pseudo-collinear set $PC(ax + by + c = 0)$, which is defined by a line equation $ax + by + c = 0$ with real constants a, b, c and the Euclidean distance d. The distance function distance$(ax + by + c = 0, p)$ calculates the normal distance of a line and a point and is defined as

$$\text{distance}(ax + by + c = 0, p) = \frac{|ap_x + bp_y + c|}{\sqrt{a^2 + b^2}} \tag{3.3}$$

[Anton and Rorres, 2010]

Figure 3.3 visualizes this prerequisite. All black points belong to the pseudo-collinear plane (light gray area) created by the threshold d and the black line. The other gray points are outliers, because they are too distant from the line.

3.4 Random Connections and Cross-Ratio

With this method we fixed issues occurring in the previous algorithm (Section 3.2). The first change is to extend the searching area for reliable points between the random combinations of the key points found by the corner detector. This increases the chance to find reliable interest points. As consequence even the points in the initial image are not exactly collinear. But that is not a problem, because afterwards for all images the points are checked for pseudo-collinearity and not collinearity.

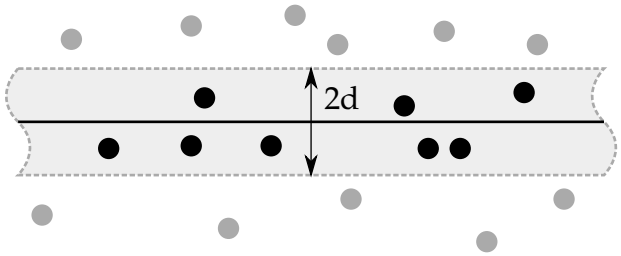


Figure 3.3: Pseudo-Collinearity; the gray area is the region belonging to the pseudo-collinear with the threshold d; all the black points are inliers and the gray ones are outliers

The distance between the additional pixels and the constructing line must be smaller or equal to the threshold d of the pseudo-collinear area. Figure 3.2b shows the difference to the old method, which can be seen in Figure 3.2a on page 15. The number of possible candidates has increased.

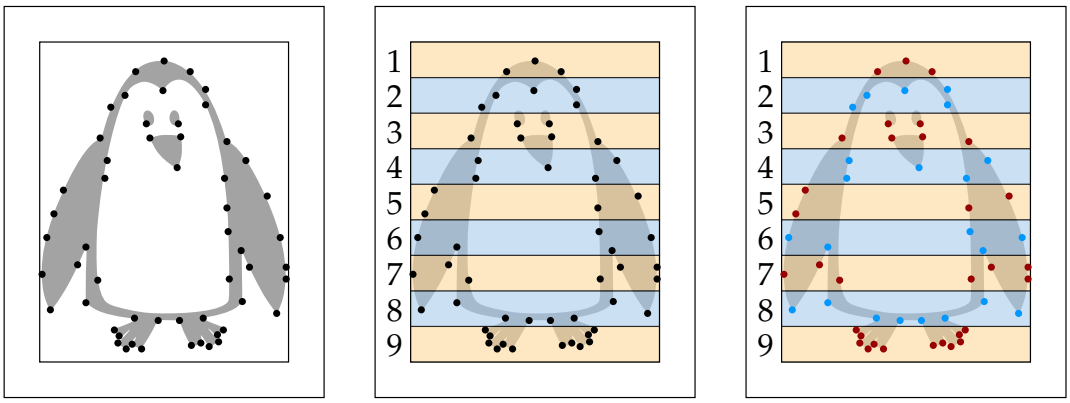
The method uses Equation (2.7) to calculate the cross-ratio. We use the distances between the four points for the calculation of the ratio. With this addition it is possible to detect unintended drifts between the four points.

But even now, after this change, the selection of the additional two points during the initialization phase yields to points, which have lower scores, than the worst scored point by the *Good Features to Track* detector. The only exception are those points, which are already points from the feature detector. Another drawback of the algorithm is the randomized selection of the point pairs. As a result the method has a highly nondeterministic behavior. Therefore the outcomes of the tracking process have a wide variation, depending on the selected combinations in the beginning.

3.5 Rectangular Templates

The previous method shows a number of disadvantages. The approach described in this section eliminates them with a new initialization procedure. This procedure has deterministic behavior.

After the corner detection the target area is separated into equal sized rectangles. There is no gap between two neighboring regions and they do not overlap too. Therefore it is possible to assign every point to such an area. These rectangles have the same height as two times the pseudo-collinear threshold d. This distance assures, that all points of the rectangle lie within its pseudo-collinear area. We call this combination of rectangles a "template". Figure 3.4 shows this basic procedure. The outer rectangle is the image and the inner the bounding box around the object of interest. The black filled circles in Figure 3.4a show the key points detected by the corner detector. After that, the image is separated into neighboring rectangles. Each section has a unique id beginning with one. In Figure 3.4b the different regions are alternately colored for better distinguishability. Every key point is located exactly in one of these planes and can be associated with it. The association is stored in a data structure. This affiliation is illustrated in Figure 3.4c by red and blue colored key points.



(a) Detect Interest Points (b) Generate Template (c) Add Points to Templates

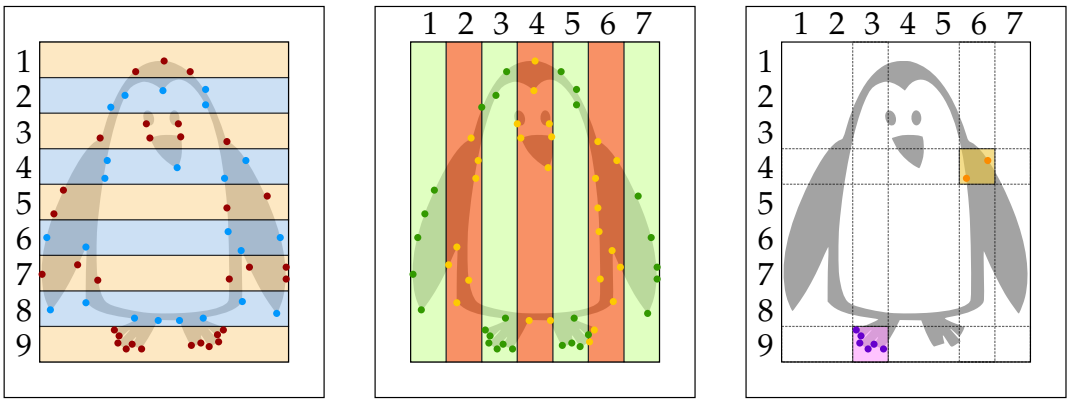
Figure 3.4: Horizontal Template; the area of the target is separated into rectangles, afterwards the points are associated with their rectangle (Penguin Image from [Pen, 2014])

In all processing iterations the points from each section build a new pseudo-collinear area and should be located within its threshold. The line belonging to the pseudo-collinear area is estimated with *RANSAC* [Fischler and Bolles, 1981]. For the next descriptions only those points are considered, which are linked to the same rectangle. This algorithm randomly picks two points to create a line and measures the normal distance from each point to the line. All points within a distance smaller or equal to the threshold d are counted as inliers. This procedure is repeated until a combination is found with no outliers, or the maximum number of repetitions is reached. That line is chosen, which has the most inliers. The outliers are removed from this pseudo-collinear plane and are not considered for the next frame.

Because only the normal distance between the points and their line of the pseudo-collinear area is measured, it is only possible to detect drifts normal to the line. Displacements in the direction of the line are not registered. Again the cross-ratio could be used to recognize these movements. In comparison to earlier discussed algorithms, not only four points build a pseudo-collinear section, but rather several belong to a pseudo-collinear area. The cross ratio is always defined by exactly four values. The question is which four points should be chosen for the cross ratio. It is possible to select them randomly. But this causes another non deterministic element in the algorithm. For that reason, we did not consider the usage of the cross-ratio in this scenario.

3.6 Two Rectangular Templates

The earlier method came up with a deterministic initialization stage. Furthermore, it simplified this step and the key points became easier to track. But one necessity of the tracker is to recognize drifts of the key points in all directions, not only normally to the line. Because all pseudo-collinears are parallel in the beginning, movements can only be detected in their normal orientation. With a second template, which is rotated about 90 degrees towards the first one, every point belongs to exactly



(a) Horizontal Template (b) Vertical Template (c) Rectangular Patch

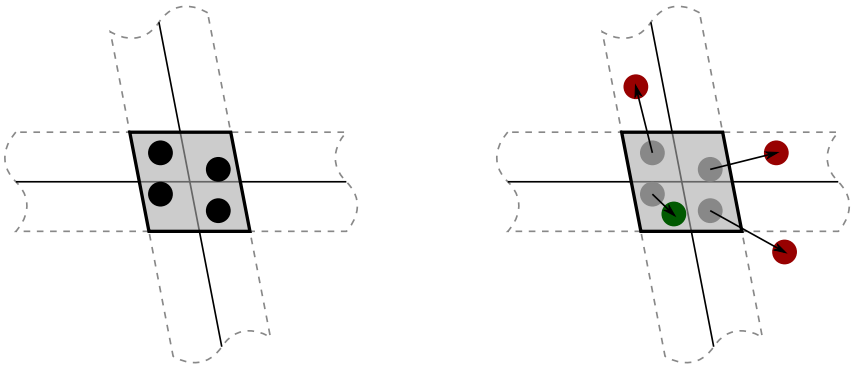
Figure 3.5: Horizontal and Vertical Template; each point is associated with a horizontal and a vertical template (Penguin Image from [Pen, 2014])

two rectangles - one rectangle from the first template and one from the other one. Through the rotation the possibility exists to detect false movements of the interest points in all directions.

The orientation of one template can be for instance horizontal. The result is that the other one must be vertically oriented. Figure 3.5a shows the template described in Section 3.5. The additional template can be seen in Figure 3.5b. It is rotated about 90 degrees compared to the other one. In Figure 3.5c an example is visualized. In this illustration the key points at the right foot of the penguin would belong to the horizontal area with the id nine and the vertical id three. These ids and key points are purple colored. The interest points at the beginning of the left wing can be associated with the horizontal template with the id four and the vertical pseudo-collinear with the id six. All of them are in an orange color tone.

After the estimation of the new position of all key points, the pseudo-collinear areas of the first template are calculated. This is the same as discussed before in Section 3.5. Supplementary the equivalent is done for the second template. After that the distances of the points to their lines are compared with the pseudo-collinear threshold d. Points which can not satisfy the threshold distance are removed from both templates and are not considered for tracking anymore.

In case that all rectangles from one template are not parallel to any section of the different template, the unwanted movement of every single point is detected in horizontal and vertical direction. The maximum allowed distance between a point and the line in a pseudo-collinear area is the threshold d. With two overlapping rectangles a point belonging to both pseudo-collinears is only allowed to stay in the intersecting area of them. Otherwise it is detected as an outlier. Figure 3.6b illustrates key points which are located in the intersection of two pseudo-collinear areas. The outlier detection of these points, belonging to both planes, and their motion vectors are visualized in the adjacent image, in Figure 3.6b. The green interest points are inliers, the red ones outliers. The allowed area is a rhombus.



(a) Intersection (b) Outlier Detection

Figure 3.6: Outlier Detection; the points are allowed to stay in the gray colored intersecting area of the pseudo-collinears; the red points on the right are detected as outliers

3.7 Find Removed Key Points

For now the number of interest points decreases over time. At each violation of the threshold distance points are removed and therefore less key points are considered for the tracking process. But there is an way with the rectangular templates to find removed interest points. Intersecting pseudo-collinear areas are used to reduce the possible search space, when removed key points should be found again. Therefore it is necessary to calculate a descriptor for each interest point during the initialization step.

This method can only be used if every single key point can be associated with at least two different templates. The intersecting area of the rectangles, which have prior contained the removed point, has to be estimated. For this area a feature detector generates possible candidates for the removed point. Then descriptors for all these points are calculated. The initial descriptor is compared against every candidate. Because of the limited search space only a minor number of comparisons exists. The number of comparisons scale linear with the number of candidates. For the calculation of the distances between the descriptors a distance function is necessary. The *hamming distance* is a suitable method for binary descriptors [Muja and Lowe, 2012]. If the distance of the best match is lower than a selected threshold, the specific point is added again to the templates and is considered for the next tracking iterations. As you can see the area is minimal, when the two templates are orthogonal orientated.

3.8 Multiple Templates

Two differently arranged templates are enough to identify wrong displacements of the interest points and the retrieval of formerly removed points. In fact this realizes the cross-ratio the aim we had in mind. But it is also possible to use more than two templates. Additional templates give further redundancy. Their benefits are discussed in this section.

Table 3.1: Template Algorithm Comparison

	Outlier Detection		Retrieve	
Number of Templates	One Direction	Both Directions	Key Points	Pseudo-Collinears
1	yes	no	yes[1]	no
2	yes	yes	yes	no
3	yes	yes	yes	yes
4...	yes	yes	yes	yes

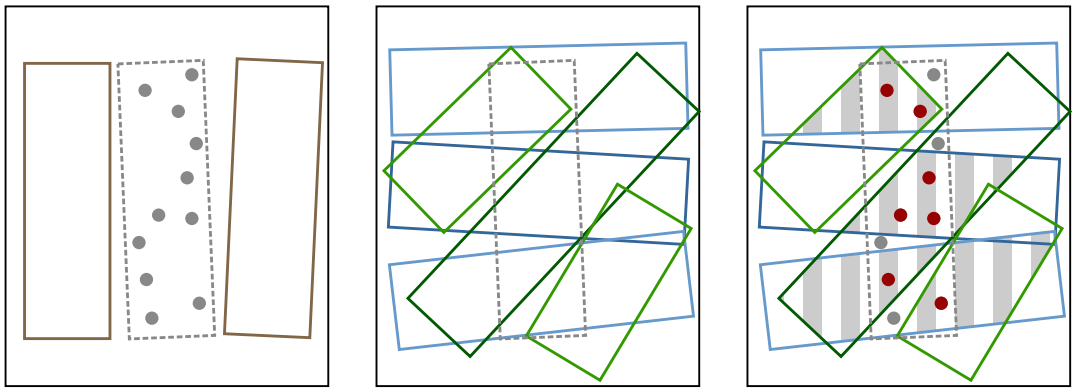
[1] yes, but the search space is the whole pseudo-collinear from one end to the other one and not only a smaller rhombus shaped area

With a third template it is possible to retrieve a whole rectangle of a template, in which all points have been removed. This is achieved by recovering the points of this area with the help of the other two templates. With at least two recovered points, the tracker is able to recalculate the position of their pseudo-collinear area. This is visualized in Figure 3.7. A fourth template does not add any additional information, it just adds more redundancy and stability.

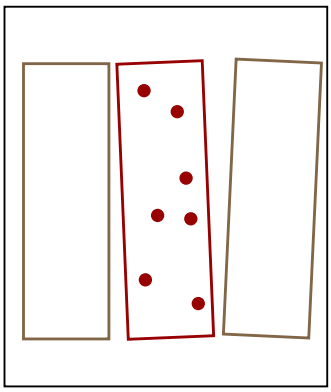
Table 3.1 shows the differences between the discussed template based algorithms.

3.9 Summary

In this chapter we explained our approaches. At first we concretized the single-target tracking algorithm from chapter one and explained it step by step. Then we discussed our first approach, where randomly chosen interest points are connected. With the help of the Bresenham's line algorithm, two additional points are added to each point connection. The collinearity criteria is checked in every frame. Afterwards we introduced the terminology pseudo-collinearity. Then we gave a description, how we extended our approach with an increased searching area during the initialization and the usage of the cross-ratio. Next we explained our rectangular template based algorithm. With this method we eliminated the indeterministic behavior of the algorithm during the initialization process. We enhanced this method by adding a second rectangular template, which is rotated against the other template. With this addition our method is able to recognize wrong displacements of the interest points in all directions. We then discussed an approach, which finds prior removed interest points in a reduced search space. At last we showed the advantages of methods with multiple templates and compared them.



(a) Lost Pseudo-Collinear (b) Other Templates (c) Find Points



(d) Retrieve Pseudo-Collinear

Figure 3.7: Retrieve Pseudo-Collinear; A vertical template is displayed in the first image in brown. The dotted gray rectangle was removed due the lost of its key points (gray points). In the second image two additional templates are drawn in green and blue. In the striped gray intersecting area of these templates, the method searches for the removed points. The obtained ones have a red filling. This is displayed in the third image. The last image shows the reinitialization of the pseudo-collinear (red rectangle) with the help of the retrieved key points.

4 Results

The last chapter described ideas how to use projective invariants in object tracking. In this part of the thesis we present the results of our experiments. For that we implemented a tracker, which uses four templates with a rotation during the initialization of 0°, 45°, 90°and 135°. The tracker is written in the programming language *Python*[1] and uses supplementary the libraries *OpenCV*[2] and *Numpy*[3].

This part of the thesis is separated into three section. In the beginning we explain the evaluation protocol and our experiment procedure. Afterwards we present the dataset and the trackers used to compare the methods. Section 4.3 forms the main part of this chapter, the experiments. In this section we measure the performance of our approach and analyze its characteristics.

4.1 Evaluation Protocol

For the measurements of the capabilities of our tracker we use the Jaccard coefficient.

$$\phi(b_T, b_{GT}) = \frac{b_T \cap b_{GT}}{b_T \cup b_{GT}} \tag{4.1}$$

b_T is the estimated bounding box of the tracker and b_{GT} refers to a manually annotated bounding box. *GT* is the abbreviation for "ground truth" and means the precise position of an object in each frame. The numerator expression calculates the intersection of the two bounding boxes and the denumerator expression the union of the areas. One benefit of this measure is, that the results are always in the interval $[0, 1]$, because the union is per definition greater or equal to the intersection. Moreover it deals accurately with scalings and translations [Hemery et al., 2007]. This method is defined on axis aligned rectangles. Therefore we calculate the smallest possible rectangle which contains the quadrilateral calculated by our tracker.

The results have to be distinguished into correct estimations and wrong ones. For that reason the results are divided into *true positives*, *true negatives*, *false negatives* and *false positives*.

- **True positives (TP)**: The output is true positive if the Jaccard coefficient ϕ is greater than a threshold τ.
- **True negative (TN)**: If there is neither an annotation in the ground truth nor an output was produced by the algorithm, the result for the specific frame is considered as a true negative.
- **False negative (FN)**: A false negative occurs if there is an entry in the ground truth but no output from the tracker is produced.
- **False positive (FP)**: The output is considered false positive if the Jaccard coefficient ϕ is smaller than the threshold τ. In addition this case leads to a false negative.

[1] `https://www.python.org/`
[2] `http://opencv.org/`
[3] `http://www.numpy.org/`

All bounding boxes generated by the tracker can be classified by Equation (4.1) and the threshold τ into *true positives* and *false negatives*.

$$\begin{cases} TP, & \phi(b_T, b_{GT}) \geq \tau \\ FN, & \phi(b_T, b_{GT}) < \tau \end{cases} \tag{4.2}$$

With these definitions the *recall* is defined as follows:

$$recall = \frac{TP}{TP + FN} \tag{4.3}$$

It can also be interpreted as *the percentage of correctly tracked frames*. The recall is our primary measure to quantify the performance of the trackers.

Visualization

For the visualization we draw success plots of the algorithms. Using the method proposed by List et al. [List et al., 2005]. On the x axis the *required recall* is assigned, on the y axis the percentages of the sequences, which fulfill the required recall are assigned. All values on both axis are in the interval $[0, 1]$ (or $[0\%, 100\%]$).

Beside these plots, which are based on the required recall, we visualize the results using the overlap measure recommended by Wu et al. [Wu et al., 2013]. These diagrams are similar to the previous ones, but on the x axis the *Jaccard coefficient* is plotted instead of the required recall. The plots show on the y axis the percentage of frames whose overlap measure ϕ is greater than the given threshold on the x axis.

Both diagrams show the performance of the trackers over all sequences in only one diagram. The best result of a tracker would be, if the line reaches the top right corner.

4.2 Dataset and Trackers

Tomáš Vojíř created a diverse dataset[4] with 77 sequences in total. From these sequences we use 25 for evaluation including partially and fully occluded humans, cars, shows, motorcycles, objects and scenes with illumination changes [Tomáš, 2013]. We use these sequences to test and compare the algorithms. Figure 4.1 shows qualitative results. The blue quadrilaterals are the results of our tracker, the yellow rectangles are their bounding boxes. Furthermore the varieties of the sequences can be seen.

We compare our algorithm with eight different trackers.

- **Consensus-based Matching and Tracking (CMT)**[5]: CMT is a new key point based tracker [Nebehay and Pflugfelder, 2014]
- **Fragments-based Tracking (FT)**[6]: FT is a basic parts-based tracker [Adam et al., 2006]
- **Structured Output Tracking (STR)**[7]: STR uses a kernelized structure [Hare et al., 2011]

[4] `http://cmp.felk.cvut.cz/~vojirtom/dataset/index.html`
[5] `https://github.com/gnebehay/CppMT`
[6] `http://www.cs.technion.ac.il/~amita/fragtrack/fragtrack.htm`
[7] `http://www.samhare.net/research/struck/code`

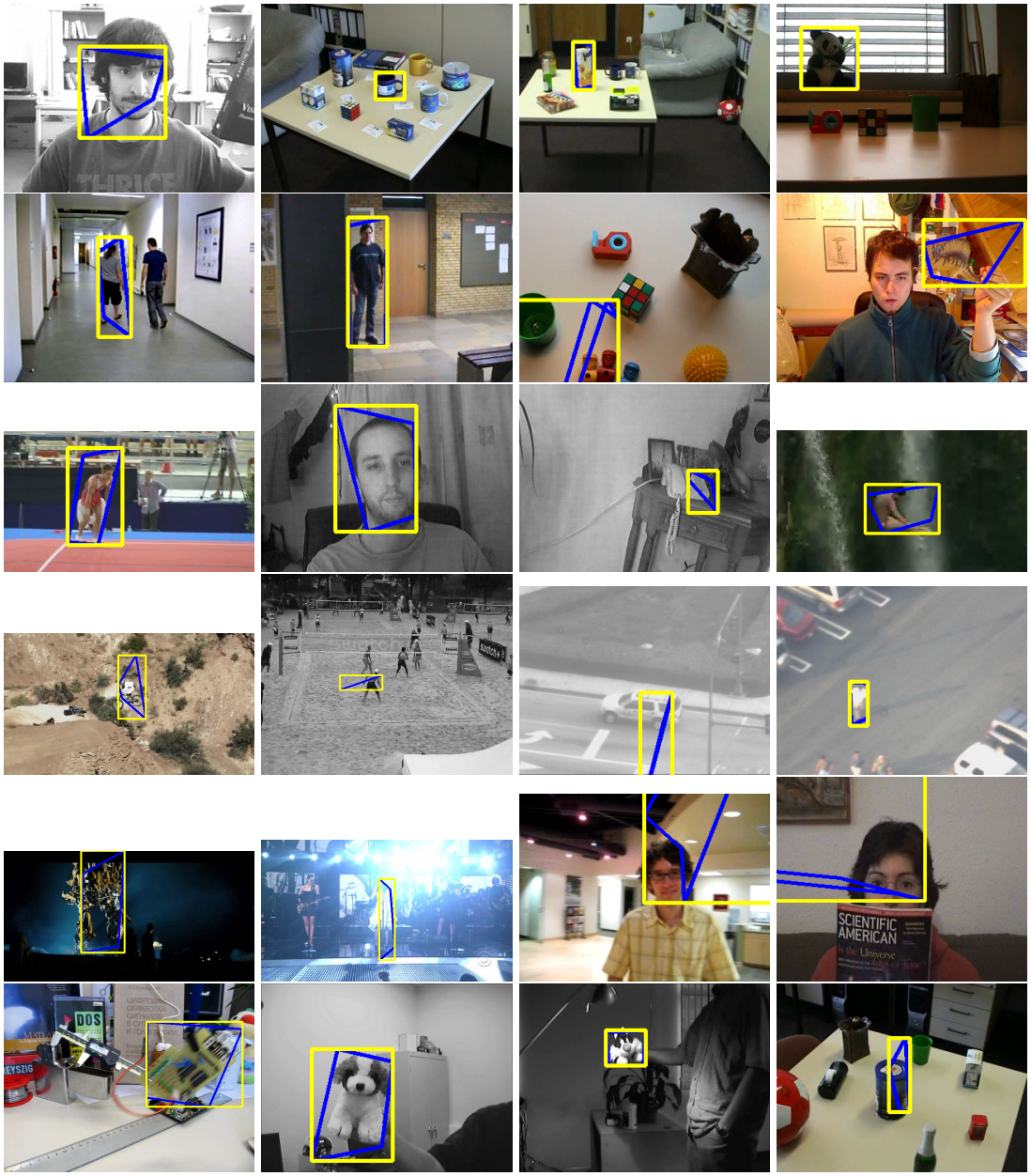


Figure 4.1: Qualitative results; the blue quadrilaterals are the results of our tracker, the yellow rectangles are their bounding boxes

- **Compressive Tracking (CT)**[8]: CT needs only a small amount of time to process frames [Zhang et al., 2012]
- **Kernelized Correlation Filters on raw pixels (KCF-2012)**[9]: KCF-2012 is a tracker using a discriminative classifier and the kernalized correlation filter for kernel regression. We use the older version of this algorithm from 2012, because it can be easily as standalone application used [Henriques et al., 2012]. There is also a newer version from 2015 [Henriques et al., 2015].
- **Discriminative Scale Space Tracker (DSST)**[10]: DSST extends the Minimum Output Sum of Squared Errors (MOSSE) tracker with robust scale estimation [Danelljan et al., 2014]; DSST is also the best performing tracker in the VOT 2014 challange [Kristan et al., 2014]
- **Mean Shift Tracker (MS)**: MS is a small tracker using mean shift and color histograms [Comaniciu and Meer, 2002]
- **Norm-Crosscorrelation Tracker (NCC)**[11]: NCC is used in the VOT-Challenge as lower bound [Kristan et al., 2014]

4.3 Experiments

In this part we compare the tracking abilities of our tracker with the methods listed in the previous section. We obtained the source code of these trackers from the internet addresses mentioned in the previous section. The only exception is the mean shift tracker: we modified an OpenCV example[12] to be compatible with the api of our experiment framework. All trackers are initialized with their default parameters.

For our tests we use a threshold of $\tau = 0.5$ for Equation (4.2), to divide the results into *true positives* and *false negatives*. Popular values for τ are 0.25, 0.5 and 0.75, which can be interpreted as low, medium and high precision requirement [Nebehay and Pflugfelder, 2014].

We use the dataset and trackers described in the previous section. Our tracker is called *Ours* in all following tables and graphics. If there is a plus sign + at the end of the name the option is enabled to find lost key points again as mentioned in Section 3.7. In both variants, our tracker uses four templates.

[8] `http://www4.comp.polyu.edu.hk/~cslzhang/CT/CT.htm`
[9] `http://home.isr.uc.pt/~henriques/circulant/`
[10] `https://github.com/gnebehay/DSST`
[11] `https://github.com/votchallenge/vot-toolkit`
[12] `http://docs.opencv.org/master/db/df8/tutorial_py_meanshift.html#gsc.tab=0`

4.3.1 Threshold Comparison

Our tracker has one primary parameter which can be adjusted, the threshold d of the pseudo-collinear templates. We determine the best value for this parameter empirically. For several different values we measure the performance of the current setting and choose the best adjustment. This adjustment is used for the comparison with the other trackers.

Table 4.1 (p. 27) shows the performance of our tracker on sequences with different thresholds. The numbers in the column header represent the threshold d of the pseudo-collinear planes and the size of the pseudo-collinear templates (Section 3.5). We use the same number for both distances. If there is a plus sign + at the end of the number, the option to find lost key points is enabled. Each row shows the recall for all trackers on one sequence. In the last line the averaged recall is written down.

The algorithm performs best with a threshold of 17. Furthermore it reveals, that it does not make a big difference if the algorithm tries to find removed points or not. The maximum difference between Ours and Ours+ for the same threshold is two percent. In 14 sequences it makes the result even worse, considering a threshold distance of 17. Through wrong estimations of the point displacement the templates degenerate. As a result the templates do not intersect each other on the same locations on the target anymore, the intersection is on a different point compared to the initialization. Thus the algorithm is not able to correctly find the lost key point or associates it with a wrong point.

For all the experiments we used the *ORB* [Rublee et al., 2011] descriptor, because its computation is efficient. The descriptor is rotation invariant in comparison to *BRISK* [Leutenegger et al., 2011], compared to *SIFT* [Lowe, 2004] it is a binary descriptor and it is free in comparison to *SURF* [Bay et al., 2008]. So *ORB* is the best solution for our experiments.

4.3.2 Tracker Comparison

For the comparisons of our tracker with the other ones, we use a threshold distance of 17, which is the best value according to the previous benchmarks. We disabled the option to find lost key points because it does not improve the results significantly, but slows the tracker down. Later benchmarks will reveal that speed is one of the biggest strengths of our method. In Table 4.2 you can see the comparison of the trackers. Our tracker is with a recall percentage of 44% comparable with *CT* (48%) and outperforms *MS* (12%) and *KCF-2012* (35%). But there is a significant gap to the best ones *CMT* (84%) and *DSST* (78%). The success plots are illustrated in Figure 4.2 (p. 28) which plots the overlap ratio, in the left image and the recall in the right.

In every sequence apart from the "board" sequence the objects of interest are not planar and therefore one prerequisite is not fulfilled. In seven sequences our approach has a recall greater than 50%. The panda in the *panda* scene is clearly not planar. But the camera does not move during the whole scene, so that is not a problem. The challenge in this scene are the illumination changes, which can be handled by all trackers except *MS* since it uses color histograms. Also the juice packaging in sequence *juice* is not a problem because trough the big distance to the camera it can be approximated as coplanar. In *shaking camera* and *person poccl* the distance from the camera to the object is also big enough, so that the shape does not

Table 4.1: Recall comparison for different thresholds d

	threshold d									
Sequence	9	9+	13	13+	17	17+	19	19+	21	21+
david short	0.21	0.22	0.25	0.24	0.29	0.24	0.22	0.31	0.22	0.21
sylvester	0.27	0.27	0.26	0.27	0.32	0.31	0.23	0.25	0.31	0.25
occl face 2	0.31	0.28	0.26	0.24	0.25	0.16	0.22	0.20	0.29	0.20
shaking camera	0.34	0.46	0.50	0.39	0.75	0.71	0.69	0.30	0.49	0.54
person foccl	0.29	0.30	0.31	0.31	0.32	0.29	0.30	0.31	0.29	0.30
singer 1	0.50	0.33	0.30	0.37	0.37	0.43	0.33	0.42	0.30	0.44
rubikscube	0.35	0.30	0.32	0.31	0.35	0.33	0.33	0.36	0.35	0.33
gymnastics	0.40	0.43	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41
cliff dive 1	0.33	0.42	0.37	0.38	0.39	0.55	0.37	0.37	0.38	0.39
volleyball	0.17	0.17	0.22	0.20	0.21	0.19	0.20	0.17	0.21	0.21
pedestrian 5	0.31	0.33	0.31	0.33	0.31	0.31	0.31	0.31	0.31	0.31
faceocc 1	0.30	0.20	0.20	0.19	0.23	0.34	0.30	0.37	0.20	0.30
mountain bike	0.35	0.64	0.24	0.42	0.46	0.44	0.52	0.37	0.34	0.30
panda	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
person poccl	0.96	0.97	0.95	0.99	0.94	0.95	0.96	0.96	0.96	1.00
dinosaur	0.22	0.16	0.15	0.17	0.27	0.25	0.15	0.16	0.17	0.16
transformer	0.32	0.26	0.30	0.34	0.23	0.32	0.29	0.31	0.31	0.33
juice	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
dog 1	0.38	0.40	0.40	0.26	0.43	0.23	0.34	0.34	0.55	0.28
car	0.22	0.20	0.21	0.20	0.22	0.22	0.23	0.22	0.23	0.23
head motion	0.41	0.27	0.40	0.68	0.52	0.23	0.51	0.53	0.56	0.45
coffee on table	0.26	0.24	0.23	0.25	0.32	0.24	0.24	0.26	0.29	0.33
person floor	0.22	0.18	0.21	0.23	0.27	0.21	0.22	0.23	0.22	0.22
board	0.22	0.55	0.44	0.25	0.59	0.52	0.49	0.35	0.46	0.58
cup on table	0.72	0.57	0.67	0.61	0.63	0.67	0.57	0.64	0.81	0.58
avg	0.40	0.41	0.40	0.40	0.44	0.42	0.42	0.41	0.43	0.41

Table 4.2: Recall comparison

Sequence	Ours	NCC	DSST	MS	KCF-2012	CMT	STR	FT	CT
david short	0.29	0.07	1.00	0.00	0.01	0.96	0.75	0.36	0.20
sylvester	0.32	0.46	0.84	0.11	0.92	0.94	0.92	0.73	0.71
occl face 2	0.25	0.96	1.00	0.76	1.00	0.95	1.00	0.60	0.95
shaking camera	0.75	0.92	0.92	0.02	0.34	0.93	0.35	0.78	0.27
person foccl	0.32	0.52	0.85	0.00	0.34	0.94	0.33	0.34	0.33
singer 1	0.37	0.29	1.00	0.00	0.00	1.00	0.28	0.28	0.22
rubikscube	0.35	0.52	1.00	0.00	0.86	1.00	0.84	0.81	0.82
gymnastics	0.41	0.36	0.43	0.28	0.28	0.41	0.49	0.42	0.38
cliff dive 1	0.39	0.71	0.59	0.00	0.05	0.76	0.62	0.18	0.59
volleyball	0.21	0.37	0.42	0.22	0.37	0.25	0.59	0.34	0.51
pedestrian 5	0.31	0.50	0.34	0.12	0.01	0.97	0.60	0.38	0.33
faceocc 1	0.23	1.00	1.00	0.00	0.05	1.00	1.00	1.00	0.55
mountain bike	0.46	0.16	0.94	0.00	0.00	0.93	0.91	0.63	0.94
panda	1.00	1.00	1.00	0.20	1.00	1.00	1.00	1.00	1.00
person poccl	0.94	0.91	0.95	0.00	0.91	0.95	0.91	0.91	0.91
dinosaur	0.27	0.07	0.37	0.00	0.01	0.23	0.24	0.09	0.12
transformer	0.23	0.15	0.40	0.00	0.01	0.54	0.53	0.51	0.40
juice	1.00	0.43	1.00	0.00	0.03	1.00	0.48	0.08	0.48
dog 1	0.43	0.60	0.98	0.11	0.70	0.98	0.67	0.63	0.56
car	0.22	0.80	1.00	0.10	0.00	0.98	0.32	0.81	0.18
head motion	0.52	0.91	1.00	0.61	0.11	1.00	0.94	0.92	0.94
coffee on table	0.32	0.23	0.46	0.00	0.18	0.97	0.22	0.24	0.18
person floor	0.27	0.60	0.31	0.00	0.57	0.49	0.34	0.79	0.19
board	0.59	0.09	0.71	0.00	0.00	0.77	0.81	0.70	0.18
cup on table	0.63	0.90	1.00	0.38	0.90	1.00	0.91	0.88	0.00
avg	0.44	0.54	0.78	0.12	0.35	0.84	0.64	0.58	0.48

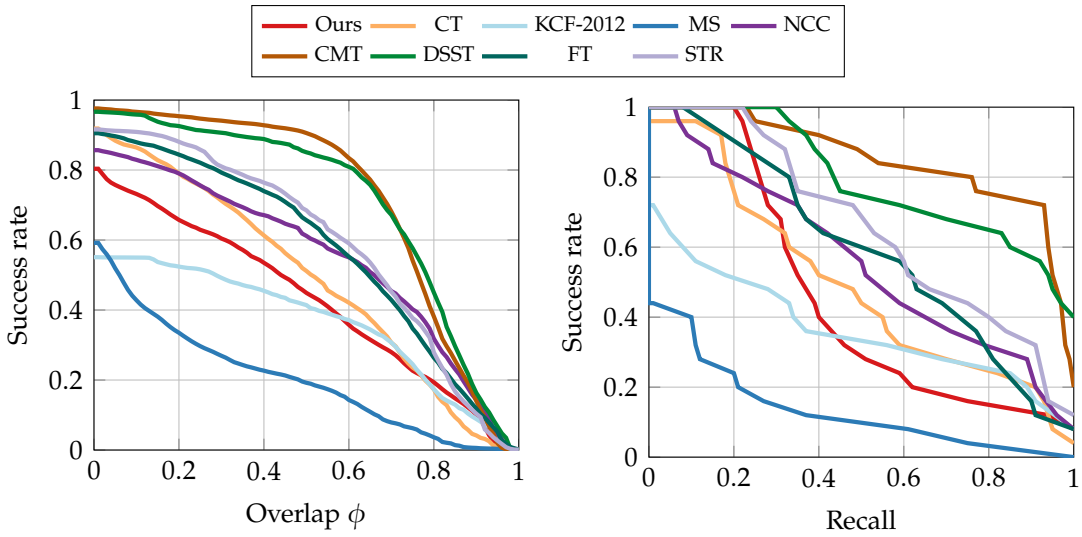


Figure 4.2: Algorithm comparison; in the left image is the success plot of the overlap measure plotted. On the right is the success plot of the recall drawn.

disturb the tracker. Another difficulty are full occlusions and overlaps. Due to the fact that optical flow is not able to handle these cases, our tracker performs worse on sequences with these events. For instance in the scene *person floor* two people cross their way and the tracked person is fully occluded. Then the movement predictions of optical flow are wrong.

4.3.3 Threshold Comparison on Short Sequences

Because other trackers like *CMT* are able to reinitialize the tracker after full occlusions and ours is not, we shortened the sequences to a maximum of 100 frames per sequence to make a fair scenario for our method. We chose to pick the first 100 frames, because analysis of the sequences revealed that full occlusions occur only after after the 100^{th} frame in all scenes.

We evaluated the best pseudo-collinear threshold distance d again. Table 4.3 shows the empirical results of these benchmarks on the shortened sequence. Now we have success rates up to 88% and have in 11 sequences a success rate of 100%. Again the additional attempt to find lost key points does not result in a significant change. The difference between the normal version and the version which tries to revive former lost key points is two percent at the maximum. Two out of five times the results are even worse. This time the tests for the best threshold result in a distance of 19.

4.3.4 Tracker Comparison on Short Sequences

In the following comparisons our tracker uses of 19 as threshold distance d for the pseudo-collinear regions. This is the best value according to the test from the last section. We have not enabled the option to find the removed points and our tracker is labeled *Ours* again. The results of this experiment can be found in Table 4.4 (p. 29). This time our tracker performs better than before and is on the fourth place out of

Table 4.3: Recall comparison for different thresholds d on shortened secquences

	threshold d									
Sequence	9	9+	13	13+	17	17+	19	19+	21	21+
david short	0.87	0.85	0.84	0.89	0.78	0.84	0.86	0.77	0.83	0.76
sylvester	0.90	0.90	0.92	0.89	0.97	0.95	0.98	0.98	0.92	0.95
occl face 2	1.00	0.95	0.94	0.98	0.99	0.92	0.93	0.94	0.97	0.97
shaking camera	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
person foccl	0.87	0.95	0.93	0.97	0.92	0.94	0.95	0.91	0.96	0.90
singer 1	1.00	1.00	1.00	1.00	0.98	1.00	1.00	1.00	0.96	1.00
rubikscube	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gymnastics	0.84	0.85	0.85	0.82	0.84	0.87	0.86	0.86	0.84	0.87
cliff dive 1	0.34	0.38	0.43	0.46	0.41	0.34	0.46	0.45	0.38	0.37
volleyball	0.92	0.87	0.96	0.98	0.99	0.98	0.98	0.99	0.99	0.99
pedestrian 5	0.69	0.67	0.67	0.71	0.68	0.68	0.68	0.68	0.68	0.68
faceocc 1	0.83	0.85	0.83	0.82	0.87	0.81	0.84	0.85	0.85	0.91
mountain bike	0.48	0.60	0.60	0.70	0.38	0.49	0.52	0.37	0.44	0.43
panda	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
person poccl	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
dinosaur	0.54	0.61	0.45	0.61	0.75	0.74	0.56	0.57	0.51	0.56
transformer	0.39	0.30	0.37	0.37	0.36	0.34	0.36	0.42	0.38	0.39
juice	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
dog 1	0.98	1.00	0.99	1.00	1.00	1.00	1.00	1.00	1.00	0.97
car	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
head motion	0.99	0.99	0.96	0.99	1.00	0.97	1.00	0.93	0.98	0.77
coffee on table	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
person floor	0.79	0.77	0.82	0.74	0.87	0.83	0.94	0.87	0.83	0.83
board	0.77	0.95	0.87	0.91	0.77	0.77	0.98	0.82	0.89	0.94
cup on table	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
avg	0.85	0.86	0.86	0.87	0.86	0.86	0.88	0.86	0.86	0.85

Table 4.4: Results on shortened sequences

Sequence	Ours	NCC	DSST	MS	KCF-2012	CMT	STR	FT	CT
david short	0.86	0.29	1.00	0.01	0.03	1.00	0.84	0.78	0.84
sylvester	0.98	1.00	1.00	0.58	1.00	1.00	1.00	1.00	1.00
occl face 2	0.93	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
shaking camera	1.00	1.00	1.00	0.15	1.00	1.00	1.00	0.98	0.72
person foccl	0.95	0.97	1.00	0.00	1.00	1.00	1.00	1.00	1.00
singer 1	1.00	0.96	1.00	0.00	0.01	1.00	0.97	0.94	0.76
rubikscube	1.00	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00
gymnastics	0.86	0.74	0.88	0.57	0.58	0.83	0.89	0.86	0.79
cliff dive 1	0.46	0.71	0.59	0.00	0.05	0.76	0.62	0.18	0.59
volleyball	0.98	1.00	0.95	1.00	1.00	1.00	1.00	1.00	1.00
pedestrian 5	0.68	0.72	0.72	0.25	0.03	0.94	0.82	0.72	0.67
faceocc 1	0.84	1.00	1.00	0.00	0.22	0.98	1.00	1.00	0.85
mountain bike	0.52	0.36	1.00	0.00	0.01	0.92	1.00	0.99	1.00
panda	1.00	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00
person poccl	1.00	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00
dinosaur	0.56	0.20	0.92	0.00	0.02	0.76	0.74	0.27	0.39
transformer	0.36	0.18	0.49	0.00	0.01	0.67	0.66	0.63	0.50
juice	1.00	1.00	1.00	0.00	0.14	1.00	1.00	0.34	1.00
dog 1	1.00	1.00	1.00	0.42	1.00	1.00	1.00	1.00	1.00
car	1.00	0.65	1.00	0.07	0.02	1.00	0.32	0.86	0.44
head motion	1.00	1.00	1.00	0.50	1.00	1.00	1.00	1.00	1.00
coffee on table	1.00	1.00	1.00	0.01	1.00	1.00	1.00	1.00	1.00
person floor	0.94	0.78	1.00	0.00	1.00	0.98	1.00	1.00	0.70
board	0.98	0.27	1.00	0.00	0.02	0.84	1.00	0.64	0.54
cup on table	1.00	1.00	1.00	0.90	1.00	1.00	1.00	1.00	0.01
avg	0.88	0.79	0.94	0.22	0.57	0.95	0.91	0.85	0.79

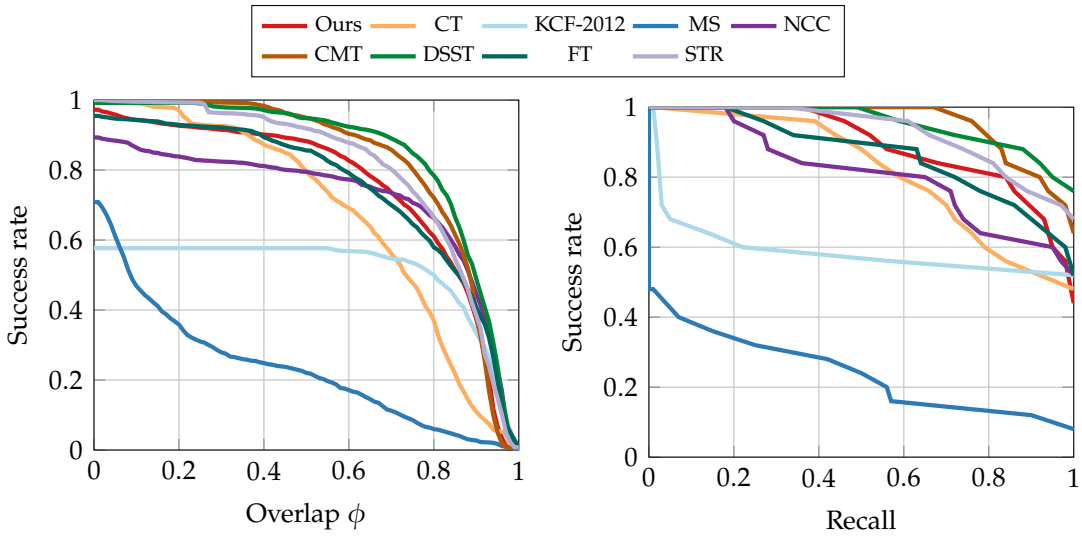


Figure 4.3: Algorithm comparison with shortened sequences; in the left image is the success plot of the overlap measure plotted. On the right is the success plot of the recall drawn. All the sequences contain only the shortened sequences from the first frame to the 100th frame.

nine trackers, compared to the seventh rank on the original sequences (see Figure 4.2 on p. 28). Our tracker has an overall recall average of 88%. Again *CMT* by Nebehay and Pflugfelder performs best with 95%. Furthermore *DSST* (94%) and *STR* (91%) perform better than our method.

In 16 sequences the difficult occlusions and perspective changes happen in images with a frame number greater than 100. For instance the crossing of the people in the sequence *person floor* happens after the 100th frame. Or in the sequence *board* the board rotates only after the removed frames. The rotation is a problem, because then the initial area of the board is not visible for some time and optical flow fails. With the reduction of the number of frames we were able to eliminate these unwanted events.

So in shorter sequences our algorithm is obviously comparable with state of the art trackers. Plots of the results are drawn in Figure 4.3. The success plots with the overlap measure are on the left side, the recall plots are on the right.

4.3.5 Timing Benchmark

Another characteristic, beside the recall, is the computation speed. We did a speed benchmark and a comparison of the results is visualized in Figure 4.4. The experiment ran on *Ubuntu Linux 14.04* with an *Intel Core i5-2430M* processor, 6 GB memory and a *NVIDIA GeForce GT 520M* graphics card. Each algorithm ran ten times under the same conditions, without any additional background processes except the operating system. The outcomes of every method got averaged to remove statistical outliers. Our algorithm performs second best with 164.43 fps. The fastest one is *MS* with 192.78 fps. But the mean shift tracker performed in both prior experiments worse than ours. There is a difference of approximately 90 fps between our tracker and *KCF-2012*, the third one, with 72.95 fps. The algorithms with the best

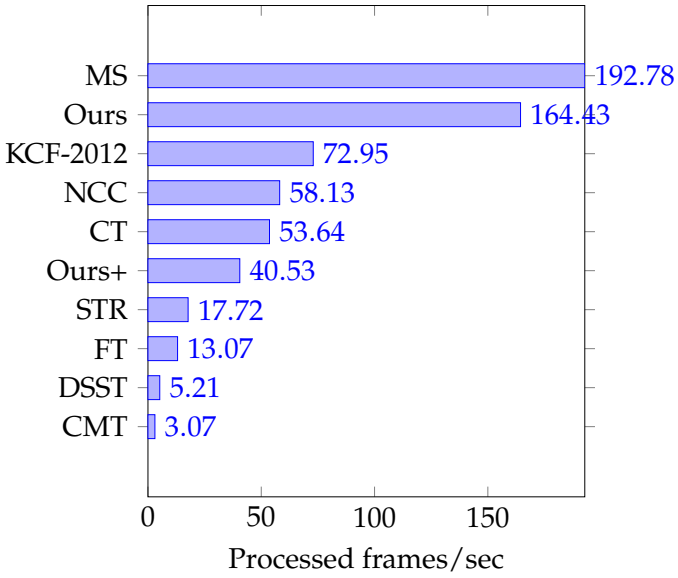


Figure 4.4: The diagram shows the averaged speed of the algorithms in frames per second

object location estimations on the long and short sequences, *CMT* and *DSST*, are the slowest ones with 3.07 fps and 5.21 fps. We benchmarked also our + tracker, but with 40.53 fps it is significantly (~120 fps) slower than our "normal" one without the enabled feature.

This benchmark reveals that the computation speed of our algorithm is one of its biggest strengths.

4.3.6 Precision vs. Speed

The results for the experiments can be summarized in a diagram. On the x axis is the speed in fps and on the y axis the average recall plotted. With the outcomes of Figure 4.4 and the results of the shortened sequences from Table 4.4 the results of every tracker can be associated with a point in this 2D diagram. We also benchmark our tracker with the enabled option to find lost key points. It uses the pseudo collinear threshold distance d 19 (see Table 4.3). There are four different quadrants in this chart. The bottom left is the worst one, there both values speed and recall are low. The preferred quadrant is at the top right, in this area the two values are great. In the top left corner the speed is slow and in the bottom right the recall values are bad. The results of this comparison are illustrated in Figure 4.5.

Eight of the trackers are positioned in the top left quadrant, also our tracker *Ours+*. There are two exceptions, the first one is *MS*, which is in the bottom right area located. The second one is our algorithm *Ours*. It is the only tracker which is located on the top right corner. Therefore our algorithm is a good trade-off between speed and recall performance for these sequences.

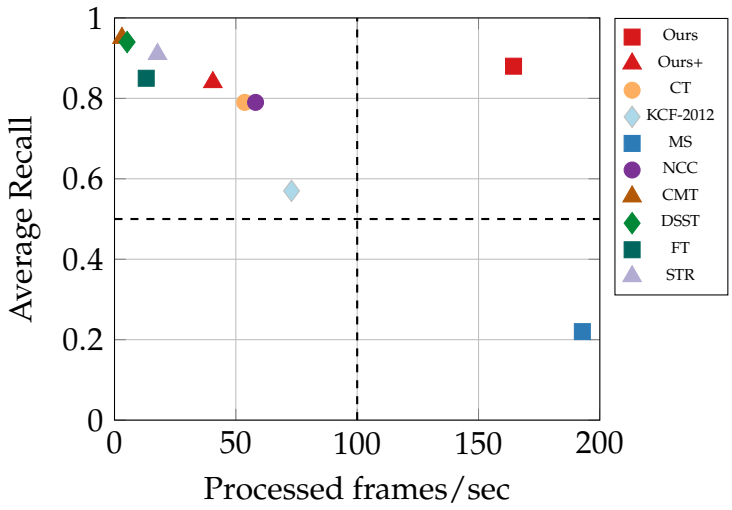


Figure 4.5: Precision vs. Speed; this plot compares the speed in frames per second with the recall precision of the methods. The preferred quadrant is the top right.

4.4 Summary

In this chapter we began with the explanation of the used measures and visualizations of the benchmark. After that we listed the trackers, which are compared with our tracker and the sequences of the dataset. We also gave qualitative results of our method. Then we determined the best pseudo-collinear threshold distance of our algorithm. Afterward we compared our method with the previous obtained threshold with the other trackers. Then we shortened the sequences to a maximum of 100 frames and determined the best threshold again. We benchmarked our and the other trackers on the shortened sequences. The best average recall of our tracker on the full sequences was 44%. It was the seventh best tracker out of nine. The best tracker was *CMT* with 84% and the last *MS* with 12%. In comparison our tracker reached 88% on the shortened sequences behind *CMT* with 95%, *DSST* with 94% and *STR* with 91%. Next we compared the computation speed of the methods and set the results of the timing benchmark in comparison with the average recall. With 164.43 fps our tracker was the fastest tracker behind *MS* which had 192.78 fps. The slowest were *DSST* with 5.21 fps and *CMT* with 3.07 fps.

5 Conclusion

In this work we elaborated a new technique to combine key point based tracking with properties of projective invariants. Therefore we experimented with several different approaches. As conclusion we dropped the idea of using cross-ratios and used the combination of two or more diverse arranged templates instead. With them it is possible to detect outliers which have moved into wrong directions. Only one template is not sufficient, because a false displacements can only be noticed in one direction. We defined a new term called *pseudo-collinearity* which is similar to collinearity but less restrict.

In terms of speed our algorithm performed second best behind *MS*. But on long sequences, with more than 100 frames, our method has not the capabilities to keep up with advanced trackers like *CMT*, which are able to handle full occlusions and are not limited to objects with coplanar surfaces. For these benchmarks we used sequences as general as possible to produce meaningful results for a variety of scenarios. This dataset contains partially and fully occluded humans, cars, shows, motorcycles, objects and scenes with illumination changes. Despite the complexity of our scenes, our pseudo-collinearity and template based tracker was able to correctly estimate the bounding box positions in seven sequences with a recall greater than 50%. To make our algorithm comparable in its use-case, we cut the scenes to a maximum of 100 frames. In this scenario our tracker was the fourth best tracker out of nine. The combination of computation speed and recall precision showed, that the approach presented in this work has the best trade-off between recall and speed.

The attempt explained in this work to regain removed key points in a reduced search space, was not able to fulfill our expectations. The processing speed was more than 120 fps slower and the maximum of the recall change was two percent.

We have ideas for potential tracker improvements. They are discussed in the next paragraphs. A necessity to compete with trackers like *DSST*, *CMT* and *STR* on the full sequences is the ability to handle full occlusions. An option would be to combine our algorithm with *CMT*, which is also key point based.

Our tracker was the second fastest trackers although it is programmed in *Python*, a programming language which is interpreted and not compiled into machine code. The computation time can be further decreased by the reimplementation of time-critical parts in *C* or *C++*.

Another weakness of our tracker was the calculation of the bounding box. The computation of the matrix for the projective transformation resulted in distorted quadrilaterals. A simpler transformation like the similarity transformation would improve the robustness if the tracking process.

With hierarchical templates the possible applications for the tracker can be extended to partially planar objects. A solution is to represent the object with pyramidal patches. The width and height of the patches in each pyramid level are reduced by a factor of two compared to the previous level. The tracker starts to estimate the new position of the patches at the lowest level. These patches vote for the position of the higher ones. Thus wrong votes by patches which cover non planar parts of the object are overruled.

Appendix A Corner Detector

For our algorithm the corner detector plays a crucial role, because it chooses the interest points which are used to calculate to object's position. We use *Good Features to Track* [Shi and Tomasi, 1994] by Shi et al. for the corner detections. In the following paragraphs we discuss this algorithm and its predecessor *Harris Corner Detector* [Harris and Stephens, 1988].

Corners are regions with a large variation in intensity along the x and y axis [Harris and Stephens, 1988]. In the following equations E describes the change in intensity for a displacement of (x, y) in all directions. The window function w can either be a rectangular or Gaussian window.

$$E(x,y) = \sum_{u,v} \underbrace{w(u,v)}_{\text{window function}} \; [\underbrace{I(x+u, y+v)}_{\text{shifted intensity}} - \underbrace{I(u,v)}_{\text{intensity}}]^2 \tag{A.1}$$

To find corners, E has to be maximized. With Taylor expansion and the prerequisite of only small shifts (x, y) the equation can be simplified to

$$E(x,y) \approx \begin{bmatrix} x & y \end{bmatrix} \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} \tag{A.2}$$

With the image derivatives I_x and I_y in x and y direction, the 2×2 matrix $\mathbf{M}$ is

$$\mathbf{M} = \sum_{u,v} w(u,v) \begin{bmatrix} I_x I_x & I_x I_y \\ I_y I_x & I_y I_y \end{bmatrix} \tag{A.3}$$

Afterwards a score is created whether to specify if a corner located in the window is a corner or not. *Good Features to Track* uses the following scoring function

$$R = \min(\lambda_1, \lambda_2) \tag{A.4}$$

Where λ_1 and λ_2 are the eigenvalues of $\mathbf{M}$. In comparison the scoring from the *Harris Corner Detector* is a bit more complex and is defined as

$$R_H = \det(\mathbf{M}) - k \cdot \operatorname{trace}(\mathbf{M})^2 = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{A.5}$$

Appendix B Ray Casting

Our algorithm can not only be initialized with a rectangle, but also with a general quadrilateral or a polygon. The corner detector calculates interest points for the whole image, but we only need the points inside the initial region. Therefore it's necessary to know which points are located in this region and which are not. To distinguish points inside and outside a axis aligned rectangle, each point has to be checked if it is located between the top left and the bottom right coordinate of the object [Shalev-Shwartz and Ben-David, 2014].

For polygons this approach is insufficient. A solution for this problem provides the *ray casting* algorithm. A ray comes from any direction outside the polygon and ends at the point. If the number of intersections between the ray and the polygon is odd then the point is located inside the polygon otherwise it is outside [Hormann and Agathos, 2001]. Figure B.1 illustrates the principle of the ray casting algorithm.

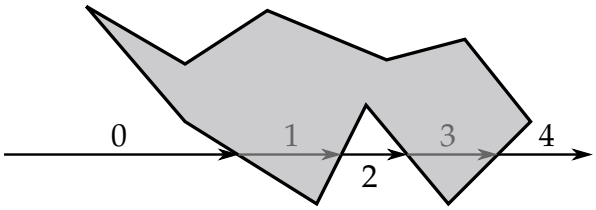


Figure B.1: Ray Casting

References

[Pen, 2014] (2014). Penguin animal. `http://clipartist.net/links/clipartist.net/penguin_animal.svg`. [Online; accessed August 14, 2015].

[Adam et al., 2006] Adam, A., Rivlin, E., and Shimshoni, I. (2006). Robust fragments-based tracking using the integral histogram. In *Computer Vision and Pattern Recognition*, pages 798–805.

[Anton and Rorres, 2010] Anton, H. and Rorres, C. (2010). *Elementary Linear Algebra: Applications Version*. John Wiley & Sons.

[Baker et al., 2011] Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M. J., and Szeliski, R. (2011). A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31.

[Bay et al., 2008] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359.

[Belot, 2011] Belot, G. (2011). *Geometric Possibility*. OUP Oxford.

[Bennamoun and Mamic, 2012] Bennamoun, M. and Mamic, G. J. (2012). *Object recognition: fundamentals and case studies*. Springer Science & Business Media.

[Bradski and Kaehler, 2008] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV, [Computer Vision with OpenCV Library ; software that sees]*. O'Reilly Media, 1. ed. edition.

[Bronstein and Semendjajew, 1979] Bronstein, I. and Semendjajew, A. (1979). *Taschenbuch der Mathematik*. BSB B. G. Teubner Verlagsgesellschaft, Nauka-Verlag, 19 edition.

[Challa, 2011] Challa, S. (2011). *Fundamentals of Object Tracking*. Cambridge books online. Cambridge University Press.

[Comaniciu and Meer, 2002] Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619.

[Comaniciu et al., 2000] Comaniciu, D., Ramesh, V., and Meer, P. (2000). Real-time tracking of non-rigid objects using mean shift. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, volume 2, pages 142–149. IEEE Comput. Soc.

[Danelljan et al., 2014] Danelljan, M., Häger, G., Shahbaz Khan, F., and Felsberg, M. (2014). Accurate scale estimation for robust visual tracking. In *Proceedings of the British Machine Vision Conference*. BMVA Press.

[Eltoukhy and Salama, 2002] Eltoukhy, H. and Salama, K. (2002). Multiple camera tracking. *Project Report*.

[Falk, 1972] Falk, G. (1972). Interpretation of imperfect line data as a three-dimensional scene. *Artificial intelligence*, 3:101–144.

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.

[Forsyth and Ponce, 2002] Forsyth, D. A. and Ponce, J. (2002). *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference.

[Gardiner et al., 1985] Gardiner, C. W. et al. (1985). *Handbook of stochastic methods*, volume 4. Springer Berlin.

[Gibson, 1950] Gibson, J. J. (1950). *The Perception of the Visual World*. Houghton Mifflin, Boston, MA.

[Hare et al., 2011] Hare, S., Saffari, A., and Torr, P. H. S. (2011). Struck: Structured output tracking with kernels. In Metaxas, D. N., Quan, L., Sanfeliu, A., and Gool, L. J. V., editors, *ICCV*, pages 263–270. IEEE.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151.

[Hartley and Zisserman, 2003] Hartley, R. I. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

[Hemery et al., 2007] Hemery, B., Laurent, H., and Rosenberger, C. (2007). Comparative study of metrics for evaluation of object localisation by bounding boxes. In *International Conference on Image and Graphics*, pages 459–464. IEEE.

[Henriques et al., 2012] Henriques, J. F., Caseiro, R., Martins, P., and Batista, J. (2012). Exploiting the circulant structure of tracking-by-detection with kernels. In *proceedings of the European Conference on Computer Vision*.

[Henriques et al., 2015] Henriques, J. F., Caseiro, R., Martins, P., and Batista, J. (2015). High-speed tracking with kernelized correlation filters. *IEEE Transactions on, Pattern Analysis and Machine Intelligence*, 37(3):583–596.

[Heuel, 2004] Heuel, S. (2004). *Uncertain Projective Geometry: Statistical Reasoning For Polyhedral Object Reconstruction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Hochberg, 1994] Hochberg, J. (1994). James jerome gibson 1904–1979. *Biographical Memoirs, National Academy of Sciences*, 63:150–171.

[Hormann and Agathos, 2001] Hormann, K. and Agathos, A. (2001). The point in polygon problem for arbitrary polygons. *Comput. Geom. Theory Appl.*, 20(3):131–144.

[Huang, 1996] Huang, T. (1996). Computer vision: Evolution and promise. *1996 CERN SCHOOL OF COMPUTING*, page 21.

[Kalal et al., 2010] Kalal, Z., Mikolajczyk, K., and Matas, J. (2010). Forward-backward error: Automatic detection of tracking failures. In *In Proceedings of the 2010 20th International Conference on Pattern Recognition, ICPR '10*, pages 2756–2759. IEEE Computer Society.

[Kalal et al., 2012] Kalal, Z., Mikolajczyk, K., and Matas, J. (2012). Tracking-learning-detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(7):1409–1422.

[Kohler et al., 2011] Kohler, J., Pagani, A., and Stricker, D. (2011). Detection and identification techniques for markers used in computer vision. *Visualization of Large and Unstructured Data Sets-Applications in Geospatial Planning, Modeling and Engineering*, 19:36–44.

[Kristan et al., 2014] Kristan, M., Pflugfelder, R., Leonardis, A., Matas, J., Cehovin, L., Nebehay, G., Vojir, T., and Fernandez, G. (2014). The Visual Object Tracking VOT2014: Challenge and results. In *Workshop on the VOT2014 Visual Object Tracking Challenge*, pages 191–217. Springer.

[Leutenegger et al., 2011] Leutenegger, S., Chli, M., and Siegwart, R. Y. (2011). Brisk: Binary robust invariant scalable keypoints. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2548–2555, Washington, DC, USA. IEEE Computer Society.

[List et al., 2005] List, T., Bins, J., Vazquez, J., and Fisher, R. (2005). Performance evaluating the evaluator. In *Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on*, pages 129–136.

[Loaiza et al., 2007] Loaiza, M. E., Raposo, A., and Gattass, M. (2007). A novel optical tracking algorithm for point-based projective invariant marker patterns. In Bebis, G., Boyle, R. D., Parvin, B., Koracin, D., Paragios, N., Syeda-Mahmood, T. F., Ju, T., Liu, Z., Coquillart, S., Cruz-Neira, C., Müller, T., and Malzbender, T., editors, *ISVC (1)*, volume 4841 of *Lecture Notes in Computer Science*, pages 160–169. Springer.

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110.

[Lucas and Kanade, 1981] Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Maggio and Cavallaro, 2011] Maggio, E. and Cavallaro, A. (2011). *Video tracking: theory and practice*. John Wiley & Sons.

[Mahmoudi et al., 2014] Mahmoudi, S., Kierzynka, M., Manneback, P., and Kurowski, K. (2014). Real-time motion tracking using optical flow on multiple gpus. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, 62(1):139–150.

[Muja and Lowe, 2012] Muja, M. and Lowe, D. G. (2012). Fast matching of binary features. In *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, pages 404–410. IEEE.

[Mundy and Zisserman, 1992] Mundy, J. and Zisserman, A. (1992). *Geometric Invariance in Computer Vision*. Artificial intelligence. MIT Press.

[Nebehay and Pflugfelder, 2014] Nebehay, G. and Pflugfelder, R. (2014). Consensus-based matching and tracking of keypoints for object tracking. In *Winter Conference on Applications of Computer Vision*, pages 862–869. IEEE.

[Nebehay and Pflugfelder, 2015] Nebehay, G. and Pflugfelder, R. (2015). Clustering of Static-Adaptive correspondences for deformable object tracking. In *Computer Vision and Pattern Recognition*, pages 2784–2791. IEEE.

[Nievergelt and Hinrichs, 1993] Nievergelt, J. and Hinrichs, K. H. (1993). *Algorithms and Data Structures: With Applications to Graphics and Geometry*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Ramachandra, 2000] Ramachandra, K. (2000). *Kalman filtering techniques for radar tracking*. CRC Press.

[Roberts, 1963] Roberts, L. G. (1963). *Machine perception of three-dimensional soups*. PhD thesis, Massachusetts Institute of Technology.

[Rublee et al., 2011] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA. IEEE Computer Society.

[Sagan, 2013] Sagan, L. S. (2013). Human outline. `https://commons.wikimedia.org/wiki/File:Human_outline.svg`. [Online; accessed August 6, 2015].

[Sebesta and Baillieul, 2012] Sebesta, K. and Baillieul, J. (2012). Animal-inspired agile flight using optical flow sensing. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 3727–3734. IEEE.

[Shalev-Shwartz and Ben-David, 2014] Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

[Shannon and Weaver, 1949] Shannon, C. E. and Weaver, W. (1949). The mathematical theory of information.

[Shi and Tomasi, 1994] Shi, J. and Tomasi, C. (1994). Good features to track. In *IEEE CVPR*, pages 593–600.

[Sittler, 1964] Sittler, R. W. (1964). An optimal data association problem in surveillance theory. *Military Electronics, IEEE Transactions on*, 8(2):125–139.

[Thota et al., 2013] Thota, S. D., Vemulapalli, K. S., Chintalapati, K., and Srinivas, P. S. (2013). Comparison between the optical flow computational techniques. *International Journal of Engineering Trends and Technology*.

[Tomáš, 2013] Tomáš, V. (2013). Dataset. `http://cmp.felk.cvut.cz/~vojirtom/dataset/index.html`. [Online; accessed September 4, 2015].

[Tuytelaars and Mikolajczyk, 2008] Tuytelaars, T. and Mikolajczyk, K. (2008). Local invariant feature detectors: A survey. *Found. Trends. Comput. Graph. Vis.*, 3(3):177–280.

[van Liere and Mulder, 2003] van Liere, R. and Mulder, J. D. (2003). Optical tracking using projective invariant marker pattern properties. In *IEEE Virtual Reality Conference 2003 (VR 2003), 22-26 March 2003, Los Angeles, CA, USA, Proceedings*, pages 191–198.

[Vojir and Matas, 2014] Vojir, T. and Matas, J. (2014). The Enhanced Flock of Trackers. In *Registration and Recognition in Images and Videos*, pages 113–136. Springer Berlin Heidelberg, Springer Berlin Heidelberg.

[Wu et al., 2013] Wu, Y., Lim, J., and Yang, M.-H. (2013). Online object tracking: A benchmark. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 0:2411–2418.

[Yilmaz et al., 2006] Yilmaz, A., Javed, O., and Shah, M. (2006). Object tracking: A survey. *ACM Comput. Surv.*, 38(4).

[Zhang et al., 2012] Zhang, K., Zhang, L., and Yang, M.-H. (2012). Real-time compressive tracking. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part III*, ECCV'12, pages 864–877, Berlin, Heidelberg. Springer-Verlag.