Masaryk University
Faculty of Informatics

Mapping 2D Skeleton Sequences from Speed Climbing Videos onto a Virtual Reference Wall

Master's Thesis

Jan Pokorný

Brno, Spring 2021

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Jan Pokorný

Advisor: RNDr. Petr Eliáš, Ph.D.

Abstract

Speed climbing is an Olympic sport with a growing interest in automatic performance analysis. We design, implement, and evaluate a pipeline to compute transformations of speed climbing videos to a reference wall. First, an object detection model is fine-tuned for the task of hold detection. Next, the detections are tracked using optical flow and associated to the reference wall using Coherent Point Drift. Finally, a sequence of absolute transformations is computed, cleaned, and smoothed. A substitution study is performed, evaluating multiple algorithm choices in each step of the pipeline. The resulting transformation sequence is compatible with any 2D skeleton representation, mapping the estimated pose to reference wall coordinates. The output is an essential step in analyzing speed climbing performances without specialized equipment.

Keywords

speed climbing, pose estimation, object detection, perspective correction, machine learning, deep learing, computer vision, motion data

Contents

1 Introduction

Speed climbing is a sport that is getting more attention due to its recent inclusion in the Olympic Games. With growing popularity, interest is emerging in automatic analysis of climber's performances, allowing for more effective training. Furthermore, this sport is a good candidate for analysis using computer vision methods. The wall layout is unified, and the climber's movement happens on a fixed plane. While automatic performance analysis is standard for many sports, the research on automatic analysis of speed climbing is still in its infancy.

This thesis is a part of a larger effort to analyze and compare speed climbing performances by machine learning, statistical, and visualization-based methods. We design and implement a pipeline that computes transformations from source video frames to reference wall coordinates (see Figure 1.1). The resulting transformations allow inter-comparability between performances shot on non-calibrated cameras.

The pipeline created in this thesis integrates with an ongoing research effort to provide a complete automated framework for analyzing and comparing speed climbing performances. Unlike existing contemporary research in speed climbing analysis [1, 2], the goal is to derive usable data from footage that can be shot "in the wild", i.e., at speed climbing competitions or arbitrary training facilities with an ordinary phone or video camera. Therefore, there is no reliance on specially calibrated cameras, drones, markers, or any other kind of physical intervention.

This is a challenging problem, given that general video footage may contain unpredictable camera movement, is shot from an unknown vantage point, and often may be low resolution. In addition, the most common way of shooting speed climbing videos is following the climber with the camera, with only a fraction of the wall visible. These factors make it difficult to determine the correspondence of the footage to the reference wall, which is necessary for comparison with performances shot in different conditions. Therefore, the framework has to rely only on visual clues in the video – specifically, the climbing holds – to minimize the influence of the mentioned factors.

In order for this to be possible using a single camera, we perform the transformations in 2D planes. Detecting an inherently 3D skeleton of the climber using a 2D pose estimator inadvertently projects the skeleton to a 2D plane of the camera frame. Thus, we assume that the camera view angle is approximately perpendicular to the wall and that the climber-wall distance is negligible. These assumptions allow us to use a 2D transformation to map the video-plane skeleton to the reference wall plane.

1.1 Contributions of this thesis

The full analysis framework is planned to operate as follows, with the topic of this thesis printed in bold. For graphical representation, see Figure 1.2.

- *Video preparation*: Videos of speed climbing performances are cut and cropped to show a single performance.
- *Pose estimation*: A pose estimation model is used to determine coordinates of the climber's specific skeleton joints.
- ***Hold detection*: Hold detection model is used to determine coordinates of dominant hand-holds in the frame.**
- ***Hold registration*: Detections are assigned to corresponding reference wall holds.**
- ***Transformation calculation*: Hold positions are used to determine the transformation of each frame onto a reference wall plane.**
- *Analysis*: The transformed skeleton data are now comparable and allow for analyses, visualizations, or further processing via the means of machine learning.

This thesis focuses on creating a pipeline with the purpose of transforming skeleton joint locations in frame coordinates to reference wall coordinates, as visualized in the Figure 1.1.

The mapping is determined by exploiting the knowledge of hold positions. Speed climbing always uses the same reference wall layout

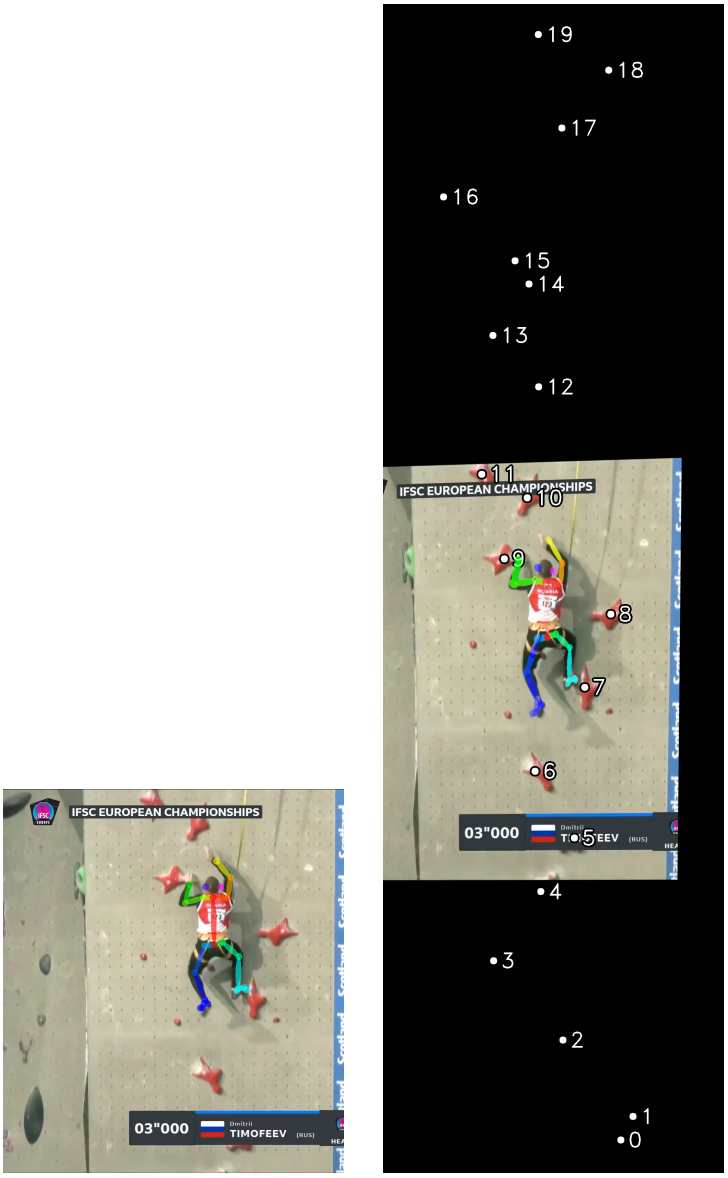
(as depicted in Figure 1.3), with a non-repeating pattern of hand-holds. The holds are an easily recognizable landmark and form a "path" that the climber follows, making them an ideal choice for alignment points. All of the hand-holds are of the same design, making it necessary to rely on relative hold positions to determine the detection-hold correspondence.

Hold positions are detected from the input video using a deep-learning object detection model. It is then determined which detections belong to which physical hold on the reference wall, using a combination of computer vision techniques. Transformations (homogeneous 3×3 transformation matrices) are then computed, cleaned, and smoothed for continuity. The pipeline's output is a series of transformations, one for each input video frame, transforming the frame to the reference wall plane. These transformations then can be applied to video (for visual evaluation) and, finally applied to skeleton data enabling their representation in uniform coordinate system.

The designed pipeline is evaluated in a substitution study, which explores the influences of various algorithm choices in the pipeline. Examined choices include different training options for the object detection model, and different approaches to finding relative transformations.

1.2 Structure of this thesis

The structure of this thesis is as follows: Chapter 2 describes the state-of-the-art work in relevant areas of speed climbing analysis, object detection, point registration, and related topics. Chapter 3 describes preliminaries and terminology necessary for the following main contributing parts. Chapter 4 describes in detail the design of the pipeline (hold detection, detection tracking, transformation calculation), and chapter 5 elaborates on the used dataset, implementation, and technical aspects. The pipeline is then evaluated in chapter 6 using a substitution study on 20 randomly selected videos.



(a) skeleton detected in frame coordinates

(b) skeleton transformed to reference wall coordinates

Figure 1.1: Objective of this work it to map estimated skeletons from video (a) into uniform coordinate space of the reference wall (b) by applying point set transformations exploiting the knowledge of automatically detected holds and wall proportions.

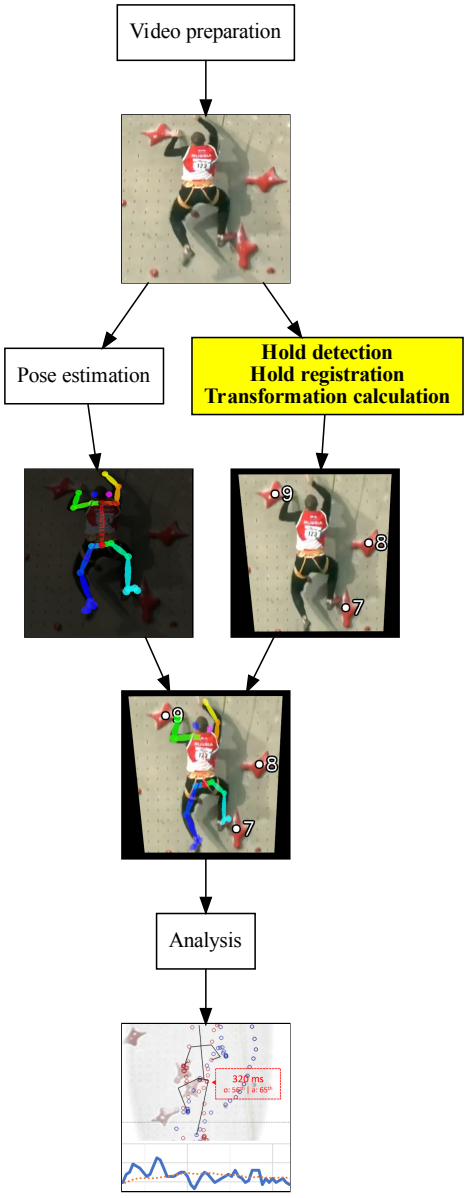


Figure 1.2: Outline of the research context, showing this thesis (highlighted node) as a part of a larger research project to perform analysis on speed climbing performances.

Figure 1.3: Speed climbing wall used in Climbing World Championships 2018. All the competitions use the exact same wall layout. Image sourced from Wikimedia Commons, author Simon Legner, licence CC BY-SA 4.0. [3]

2 Related work

While computer vision and deep learning are already prevalent in sports analysis [4], automated analysis of speed climbing is an emerging field. This chapter describes the state-of-the-art approaches in speed climbing analysis, as well as other fields relevant to this thesis (pose estimation, object detection, point set registration, and optical flow).

2.1 Speed climbing analysis

Since the Summer Youth Olympic Games 2018 in Buenos Aires, Argentina, Speed climbing is an Olympic sport and will be included in the upcoming Tokyo 2021 Olympic games. This has kindled research interest in the subject in recent years [5, 6, 7].

Existing approaches to speed climber motion tracking use specialized cameras and markers to capture the climber's position.

Reveret et al. [5] use a pair of drones that synchronize their movement with the wall using a prepared 3D scan to record a video of the climber's ascent. They then show that this approach can track a marker placed on the climber, approximating their center of mass. Alternatively, a method is proposed where a 3D hull of the athlete is approximated using a machine learning model trained using 68 cameras.

Legreneur et al. [6] use a camera stabilized using a 3D axis stabilization platform (DJI Ronin-M), placed in a specific position relative to the wall. A marker placed on the climber's body was tracked and interpreted as an estimation of the climber's center of mass.

Iguma et al. [7] research the general case of sport climbing analysis. Their research relies on a combination of 3D motion tracking using markers on climbers and force tracking using sensors placed in holds. This approach thus uses not only specialized cameras and tracking markers but also a modified wall.

No current methods for speed climbing analysis operate on standard video footage of the event alone, requiring videos captured using specialized equipment or using a marker placed on the climber.

The pipeline designed in this thesis does not have such limitations, allowing it to operate on historical footage. It has, however, different limitations compared to the two approaches outlined above. Most notably, it is 2D-only and has inherently lower accuracy given the projection to 2D.

2.2 Human pose estimation

Human pose estimation is a problem of extracting joint coordinates of a virtual skeleton corresponding to a human body from video. The state-of-the-art solutions are based on machine learning keypoint detection models.

While some approaches extract 3D coordinates from 2D video [1], they are in practice usable only on videos of usual circumstances (walking, running), and in our experiments did not perform well on speed climbing videos, as seen in Figure 2.1.

For this reason, it was decided to focus on 2D pose estimation methods, like OpenPose [8], which is demonstrated in Figure 2.2. The skeleton in OpenPose consists of 18 points, but other pose detection frameworks may use a different skeleton model. The pipeline created in this thesis does not make any assumptions about the skeleton model used, as the output transformation can be used to transform any set of points.

While this thesis does not directly work with pose estimation models, pose estimation is a core part of the broader research context (as outlined in Figure 1.2). Therefore, the pipeline in this thesis is designed with this usage in mind while not laying further assumptions on the pose estimation model used.

2.3 Object detection

Object detection is a problem of identifying physical objects in a photograph and determining their type and position. State-of-the-art object detection models are usually based on convolutional neural networks. The most notable object detection models are YOLO [2], RetinaNet [9], and SSD [10], variant of which was chosen for this thesis due to its low memory footprint.

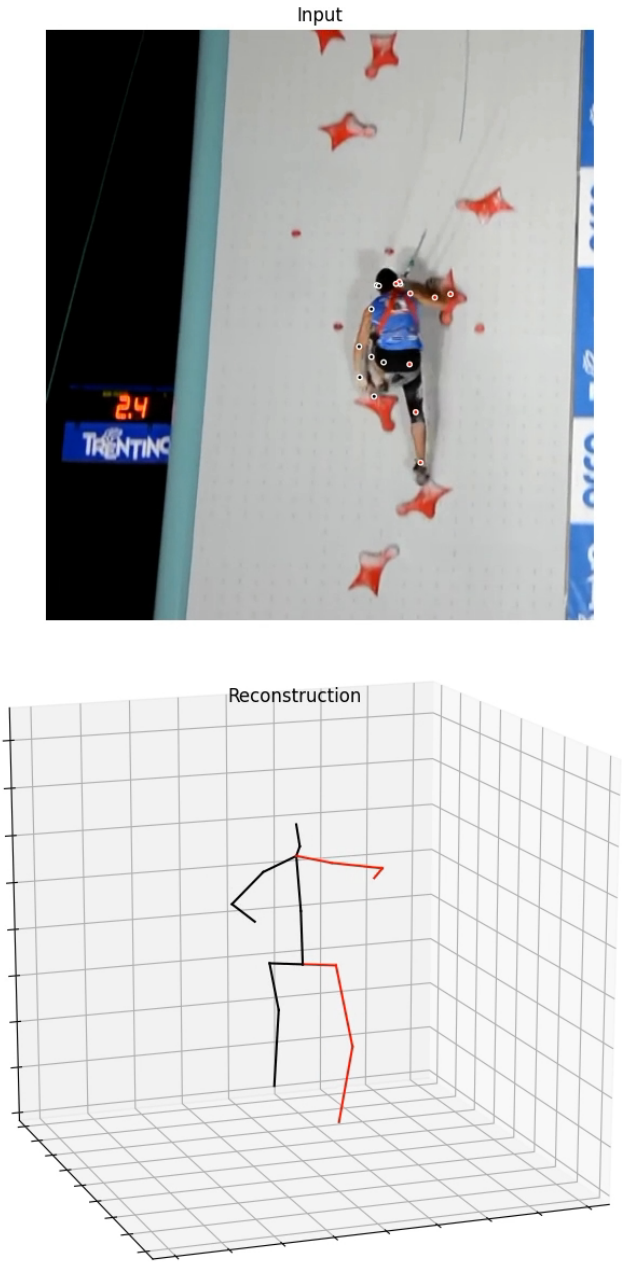


Figure 2.1: Climber's 3D pose as extracted using VideoPose3D [1]. While the 2D keypoints in the image are estimated correctly, the 3D reconstruction is obviously lacking.

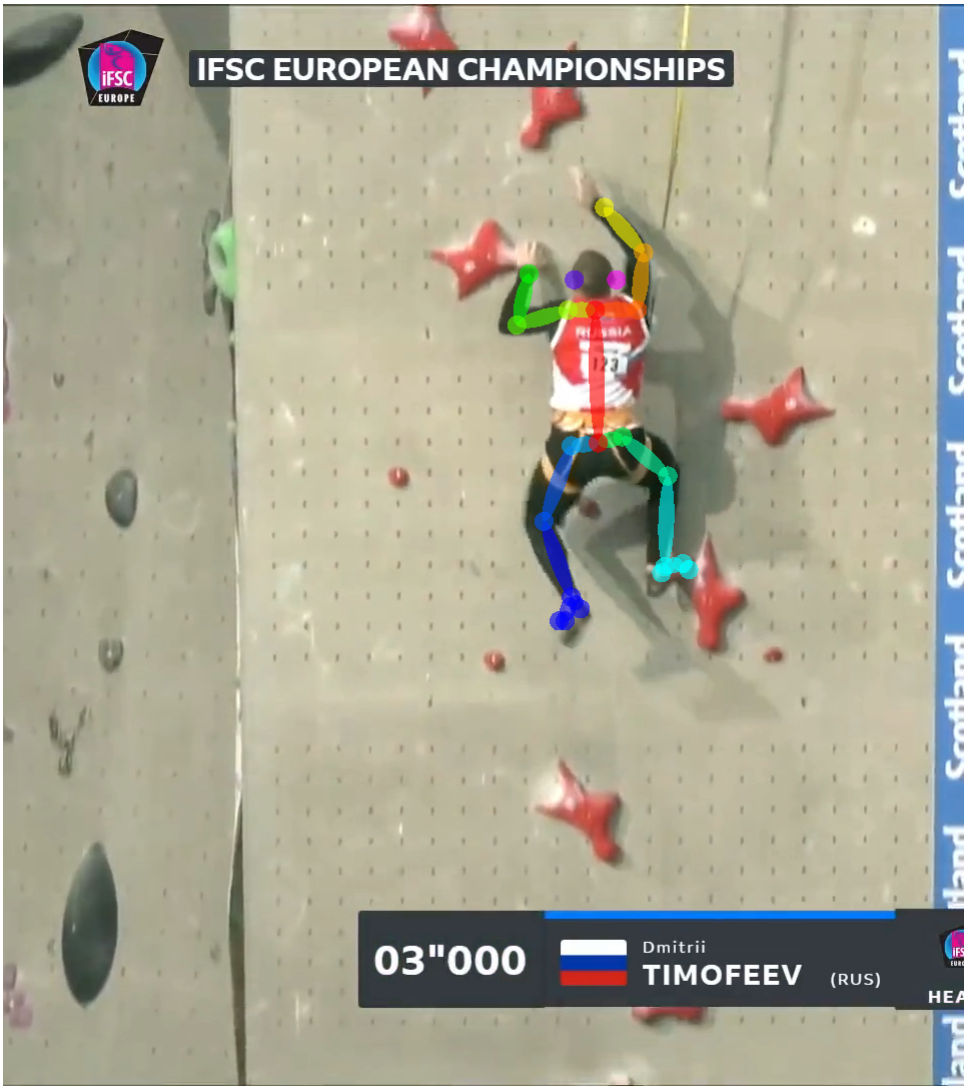


Figure 2.2: Visualisation of climber's 2D pose as extracted using OpenPose [8], consisting of 18 points.

An input of a convolutional object detection model is a square raster image (size depending on the model). The outputs are object detections, usually represented by bounding boxes (coordinates of the smallest box framing the detected object).

Implementations of different models are often grouped in frameworks. Using such frameworks gives the ability to evaluate various models and to easily switch to a better model in the future. For this thesis, TensorFlow 2 [11] Object Detection API was chosen due to its extensive configuration, large model zoo, and focus on inference.

There are many high-quality pre-trained model weights published. However, the models are trained on a limited set of object classes, "speed climbing hold" not being covered. It is still possible to take advantage of the published pre-trained models, though, using a technique called *transfer learning* (or *fine-tuning*).

The book Deep Learning by Ian Goodfellow et al. [12] describes transfer learning as "the situation where what has been learned in one setting [...] is exploited to improve generalization in another setting". In the case of this thesis, a model trained to detect everyday objects (people, cars, animals, ...) is exploited to detect a new class of objects: speed climbing holds, using just low hundreds of training examples and few hours of GPU time.

2.4 Finding a transformation between sets of points

One problem, occurring multiple times within the designed pipeline, is finding a homography transformation matrix between two planes while knowing a corresponding set of points in both planes – that is, each point in one plane has a known counterpart in the other. A point-wise matching (pairing) between the two sets may or may not be known. This section discusses the approaches employed by this thesis.

2.4.1 Paired sets of points

In an ideal case, finding the transformations for paired sets of points would mean solving the system of equations obtained for each pair of points from the following transformation equation:

$$H \begin{pmatrix} p_0 \\ p_1 \\ 1 \end{pmatrix} = \begin{pmatrix} cp'_0 \\ cp'_1 \\ c \end{pmatrix} \tag{2.1}$$

However, when operating with inexact real-world data, the problem comes out to minimizing the transformation error. In this case, the problem is to find such a transformation matrix that minimizes the sum of squared differences between the mapped source points and corresponding destination points.

The problem can be further extended by allowing outlier pairs. These can be detected using methods like random sample consensus (RANSAC) [13], which ensures that instead of fitting the transformation to the whole set, some samples are removed in order to minimize the projection error of the rest.

2.4.2 Unpaired sets of points

The problem of finding a transformation aligning two sets of points where the correspondence is not known is called "point set registration" [14]. In addition, some variants allow outliers (points without a counterpart) in one or both sets as a further complication.

Since this is a much more complex problem than the paired case, the typical approach for computer vision tasks – where the point sets usually correspond to an image – is to exploit image data for finding the point set correspondence. This approach is called "point feature matching" [15], and consists of computing a descriptor for each point using local image data – usually an n-bit number – and then pairing the points according to the local descriptors. A robust algorithm for finding a transformation between sets of paired points is then used.

However, the feature matching approach is not suitable for problems where there is no image data to back the point sets – as is the case with the topic of this thesis, where the reference wall is an abstract concept without a concrete image representation.

For the general case of registration of unpaired point sets, there are probabilistic algorithms based on point density. One such algorithm, called "Coherent point drift", was proposed in 2010 by Myronenko and Song [14]. This algorithm is employed by this thesis.

2.4.3 Associating aligned sets of points

When given two aligned planes with sets of points, assigning the correspondence between the point sets is often necessary. A functional method uses an algorithm for linear assignment [16] on a distance matrix. In order to not "over-assign", a maximum distance threshold can be added by appending a same-sized matrix full of the threshold distance to the distance matrix, allowing the points to be assigned to the "dummy" points, and interpreting those assignments as "no assignment". This method, further called "linear assignment with threshold", is employed by the selected pipeline.

2.5 Optical flow

Optical flow refers to an array of 2D displacement vectors, representing movement between two frames of a video. Optical flow can be used for object tracking by following the displacement vectors across video frames.

A standard algorithm for estimating optical flow, employed by this thesis, is the Lucas-Kanade method [17]. This method assumes that for each tracked point, some neighborhood moves uniformly with the point. Figure 2.3 shows an example of this technique.

Due to optical flow operating on pixel-level, micro-deviations (like video compression artifacts) may cause errors. It is thus usual to combine optical flow-based tracking with other techniques.

Figure 2.3: Using optical flow to track moving objects. Image sourced from OpenCV documentation [18].

3 Preliminaries and definitions

3.1 Speed climbing sport

Speed climbing is a performance sport where individuals compete to climb a 15 meters tall wall with carefully placed holds. The run starts when the starting device plays a sound and ends when the climber touches the end device at the top of the wall.

The run may also end unsuccessfully by either a fall, or a false start. This thesis only considers successfully finished runs.

According to IFSC, the current world record is 5.48 seconds, held by an Iranian climber Reza Alipourshenazandifar.

3.1.1 Reference wall

Speed climbing uses a standardized wall across all competitions, with the exact shape, placement, and even the color of the holds prescribed in the documentation published by the International Federation of Sport Climbing (IFSC) [19]. The official reference wall blueprint published by the IFSC is reproduced in Figure 3.2.

The wall has a slight incline of 5 degrees but is otherwise planar. The reference wall used in this project is an approximation obtained visually from the published schematics. Given the resolution, holds may be displaced by at most 4 mm.

3.1.2 Hold description

The speed climbing wall contains two types of holds: large hand-holds with a specific shape and small foot-holds. Since the large hand-holds are easily identifiable by their specific shape and color and occur along the entire length of the wall, it was decided to use them as reference points for computing the transformation. Thus, the pipeline uses only the large hand-holds (see Figure 3.1), and this text will refer to them as “holds”.

Figure 3.1: Closer look at speed climbing holds. Image by Hans Braxmeier from Pixabay, under a permissive licence.

3.2 Pipeline input and output

The pipeline's input is an MP4 video, which is then represented in-memory as a sequence of RGB raster frames. The pipeline's output is a 3×3 transformation matrix for each frame, transforming it into the reference wall plane. This transformation can then be easily applied by equation 3.1 to transform any set of 2D points, including 2D skeleton data[1] extracted by a pose estimation model, as outlined in Figure 1.2.

3.3 Glossary

Detection is an instance of a presumed hold position extracted from a video. It is represented by a **detection box** (represented as a 4-tuple of coordinates `(top, left, bottom, right)`) or **detection point** (center of the detection box, represented as a pair of coordinates `(x, y)`). Detections can be **associated** (or **tracked**) across frames to create **detection tracks**, which refers to determining which detections from neighboring frames correspond to the same real-world hold.

Video is as described in section 5.1. **Frame** refers to the raster data of a specific video frame, as well as the associated detections.

Reference wall is a set of hold positions in a 2D plane, representing the real-world speed climbing wall.

1. Skeleton data from various pose estimation frameworks are usually represented as an ordered set of keypoints in frame coordinates.

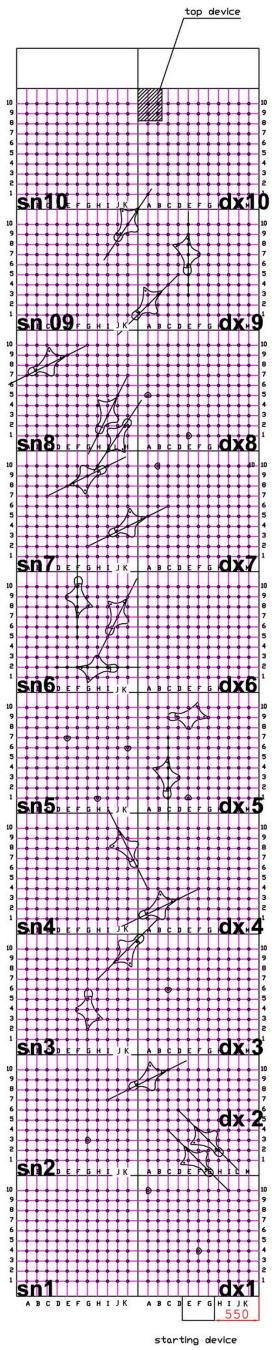


Figure 3.2: Reference blueprint of the speed climbing wall, as published by the IFSC [19].

Transformation is a 3×3 transformation matrix that can be applied to 2D points or combined. In the pipeline, each transformation is associated with a frame. **Relative transformation** is a transformation representing the camera movement between two frames. **Unrolling relative transformations** means computing a combination of all previous relative transformations up to the current frame and then projecting all detections onto a common plane. **Absolute transformation** is a transformation between a frame and the reference wall.

3.4 Transforming points using a homography transformation matrix and the influence of degrees of freedom

This thesis works with 2D points corresponding to a plane in a 3D space – the real world. One such plane is the plane of the speed climbing wall, where positions of holds can be described by their 2D coordinates relative to the wall. Another plane is the video frame, where positions of holds can be described by their 2D coordinates relative to the camera view. When there is a correspondence between two 2D planes, as in the two examples presented, it is beneficial to describe the relative difference of the two planes, allowing one to transform point coordinates in one plane to point coordinates in the other. This is achieved by using a **homography transformation matrix**, which (for 2D planes) is a 3×3 matrix H that is used to transform 2D point coordinates p to p' as follows (where c is some non-zero constant):

$$H \begin{pmatrix} p_0 \\ p_1 \\ 1 \end{pmatrix} = \begin{pmatrix} cp'_0 \\ cp'_1 \\ c \end{pmatrix} \tag{3.1}$$

This type of transformation is a homomorphism, and for real-world camera transformations, also an isomorphism – meaning that H^{-1} can be used for an inverse transformation.

8 degrees of freedom

The homography transformation matrix, of the size 3×3, has only 8 degrees of freedom, since all the matrices cH for all $c \in \mathrm{R}, c \neq 0$ represent the same transformation. It is thus usually normalized so $h_{33} = 1$, and thus the **full homography transformation matrix**, also called the **perspective transformation matrix**, has the following form:

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

Since a perspective transformation matrix has 8 degrees of freedom, it is possible to compute it from 4 pairs of 2D points (4×2 scalar coordinates). Conversely, it is possible to compute a homography transformation matrix with fewer degrees of freedom from fewer pairs of points.

6 degrees of freedom

A limited homography transformation matrix with 6 degrees of freedom representing translation, scaling, rotation, and skew is called an **affine transformation matrix**. It has two additional elements fixed: $h_{31} = h_{32} = 0$, and thus has the following form:

$$\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

4 degrees of freedom

Further limiting the transformation matrix to 4 degrees of freedom, representing translation, uniform scaling, and rotation, creates a **partial affine transformation matrix**, also often called a **rigid transformation matrix**. It has two additional elements fixed: $h_{21} = -h_{12}, h_{22} = h_{11}$, and thus has the following form:

$$\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ -h_{12} & h_{11} & h_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

2 degrees of freedom

Limiting the transformation matrix to just 2 degrees of freedom representing translation (**translation transformation matrix**) encodes the translation vector in h_{13} and h_{23}:

$$\begin{pmatrix} 1 & 0 & h_{13} \\ 0 & 1 & h_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

Even though translation can be represented simply as vector addition, it is beneficial to represent it in the same format as the more complex transformations.

4 Designing the data pipeline

This chapter describes the data pipeline for mapping speed climbing performance videos to a reference wall. While this chapter offers an algorithmic view, chapter 5 offers more details on the specific implementation.

Design and algorithm choices made in this chapter were based on iterative testing on a subset of dataset videos. The choices are then manually evaluated on a different set of videos and shown to be ideal in chapter 6.

4.1 High-level overview

An input of the data pipeline is a video capturing a single speed climbing performance, as described in section 5.1. A trained object detector is used to detect holds in each frame. The detections are then associated across frames using optical flow. Detection pairs are used to estimate relative transformations between neighboring frames. The relative transformations are then used to project detections from all frames to a common plane. The projected detections form clusters. The cluster-hold correspondence is then obtained by running a point registration algorithm, associating detection clusters with specific holds on the reference wall. The perspective transformation matrix from the video frame plane to the reference wall plane is then computed for each frame. The resulting sequence of transformations is finally smoothed using cubic spline regression. The output is a sequence of transformations that, applied to the input video frame-by-frame, maps the video frame plane to the reference wall plane.

4.2 Hold detection

We use a pre-trained object detection model and fine-tune it for the hold detection task using transfer learning. A state-of-the-art object detection model created by Sandler et al. [20] called "SSD MobileNet V2 FPNLite 320x320" was chosen for this task. The specific model was

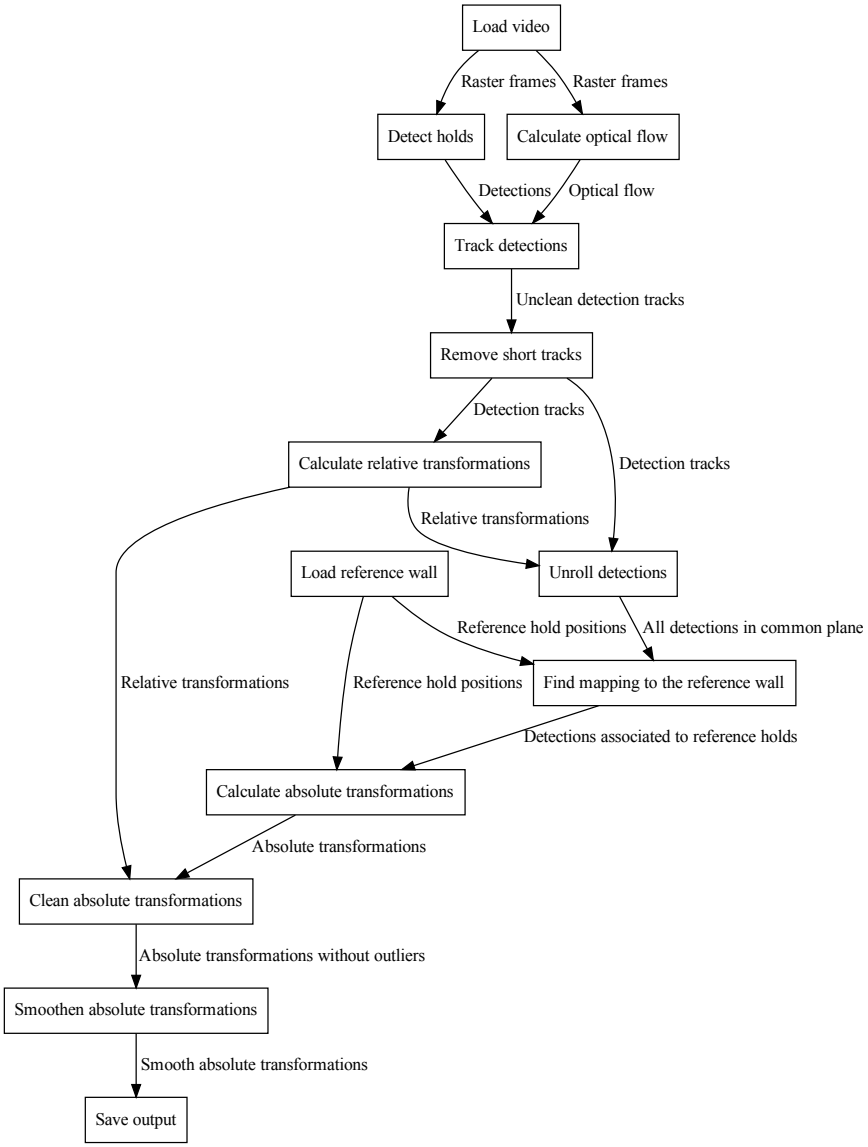


Figure 4.1: An overview diagram of the pipeline. Nodes represent pipeline steps. Edges (labeled to the right) represent data.

chosen due to its small memory footprint, allowing it to be fine-tuned even on consumer hardware[1].

The training used the default configuration for the "SSD MobileNet V2 FPNLite 320x320" model with a few changes. Due to hardware reasons, the batch size needed to be set to 32 (down from 128). The learning rate was lowered to compensate for the loss of accuracy due to the smaller batch size. Additionally, data augmentations of random horizontal and vertical flips, rotations, cropping, padding, and color distortions were added to the training pipeline to ensure robustness even with the small dataset.

4.2.1 Expected errors

Since the videos follow a climber as they climb the wall, it is expected that some holds will be temporarily covered by the climber's body, resulting in "false negative detections". In addition, overlay graphics can partially occlude holds, and the bounding box can be distorted by the hold entering or exiting the frame. Furthermore, both false positives and false negatives may be expected in any machine learning approach. Thus we design the pipeline to be resilient to both types of errors.

4.2.2 Representing detections

Standard object detectors work with bounding boxes – detections are represented by coordinates of the two diagonal corners of the smallest rectangle fully containing the detected object. However, the bounding box corner coordinates do not correspond to any physical point on the wall and are view-dependent since the bounding box is always axis-parallel to the frame. Therefore, to minimize distortion and simplify further pipeline steps, it was decided to represent hold positions using a single coordinate, corresponding to the middle point of the hold's bounding box in the reference wall plane. This point is from now on referred to as "detection point".

---

1. Only consumer hardware was available. Although consumer GPUs possess enough processing power for complex neural networks, most of them are limited by their VRAM capacity.

4.2.3 Detection threshold

With each detection, the model assigns a confidence percentage that this detection is correct. It was chosen to include all detections with confidence $> 50\%$, since, in practice, the following pipeline steps work better with more detections, even when some spurious detections are present.

4.3 Detection tracking

"Detection tracking" refers to the process of determining the relative change in detected hold positions between two captured frames and associating the hold detections between frames that belong to the same physical hold.

4.3.1 Using optical flow to associate detections

The previous step in the pipeline yielded the center points of detected holds. These detection points are now used in optical flow calculations, yielding projected positions of current frame detection points in the next frame.

The algorithm used is Lucas-Kanade feature tracker [17], which is a de-facto industry standard way of estimating optical flow. The algorithm's input are two consecutive frames from a video, capturing a slight movement between them and a set of points to be tracked, representing features in the first image. The algorithm's output is an estimation of the positions of the features in the second image, given pixel displacement between two images. It is assumed that the displacement is approximately constant in some neighborhood around a feature. This condition is satisfied most of the time in the input videos[2], given that the tracked features are the center points of detected holds. See Figure 4.2.

All the holds in the frame are expected to translate by roughly the same vector – perspective/rotational change between two frames is usually minimal. Therefore, the deltas of current and projected detec-

2. One situation where this constraint is not satisfied is when the climber's body covers the center of the hold between frames.

Figure 4.2: Associating detections through optical flow. Red dots represent detections in previous frame, green dots detections in the current frame, blue dots optical flow estimation of the previous frame detections.

tion points are averaged to obtain an estimated translation velocity of the scene in the frame to minimize error.

The translation velocity is then added to each detection point to obtain a set of projected positions, to which the detection points from the following frame are associated using linear assignment with a threshold (see section 2.4.3).

Repeating this process for all pairs of frames creates "tracks" of detection points that can be followed across frames.

Detections whose tracks are shorter than a specified threshold can then be removed, as they more likely represent spurious detections. In the selected pipeline, detections with tracks shorter than 3 were removed.

4.3.2 Computing relative transformations between frames

The associated detections are now used to calculate relative transformation between frames. A standard algorithm as described in 2.4 is used to determine the transformation between two sets of points – detection points in the current frame and the corresponding associated detection points in the next frame. This method leads to a more precise estimation of frame movement than the "frame velocity" calculated in the previous step.

4.4 Detection registration

"Detection registration" refers to the process of mapping detections to holds on the reference wall.

4.4.1 Unrolling relative transformations

The relative transformations obtained in 4.3.2 are now used to project detection points onto a common plane. To each frame's detection points, a transformation created by accumulating all previous relative transformations is applied.

As this process is prone to even minor errors in computing the relative transformations, it is beneficial to compute the relative transformations with few degrees of freedom – lowering overall accuracy while also minimizing the chance of errors.

The projection step creates a common plane, where detections from multiple frames group into clusters, representing the same physical hold.

4.4.2 Associating clusters with the reference wall

Detection clusters are positioned in a pattern that resembles the reference wall holds but in a different plane. Furthermore, there might be deviations to the pattern caused by imprecise calculation of the relative transformations between frames, and there may be both spurious points and missing ones.

The problem of finding a transformation between two sets of points with these constraints is called a *point-set registration problem*. A probabilistic algorithm called "Coherent point drift", proposed in 2010 by Myronenko and Song [14] is used to calculate an approximate affine transformation between the detection plane and the reference wall plane (see Figure 4.3).

Since Coherent point drift takes only point density into account and does not attempt to perform point associations, it is not necessary to perform any form of clustering on the detection plane.

The transformed detections in each frame are then matched with reference wall holds using a linear sum assignment algorithm with a threshold (see section 2.4.3). This ensures that the sum of errors between transformed detections and matched reference wall holds is minimized.

The matching reference wall holds are then associated with all detections in the corresponding clusters.

The method is visualised on Figure 4.3.

4.5 Obtaining the final transformations

In the final step of the pipeline, transformations of frames to the reference wall are computed from the detection-hold associations made in the previous step. These transformations are referenced as "absolute transformations".

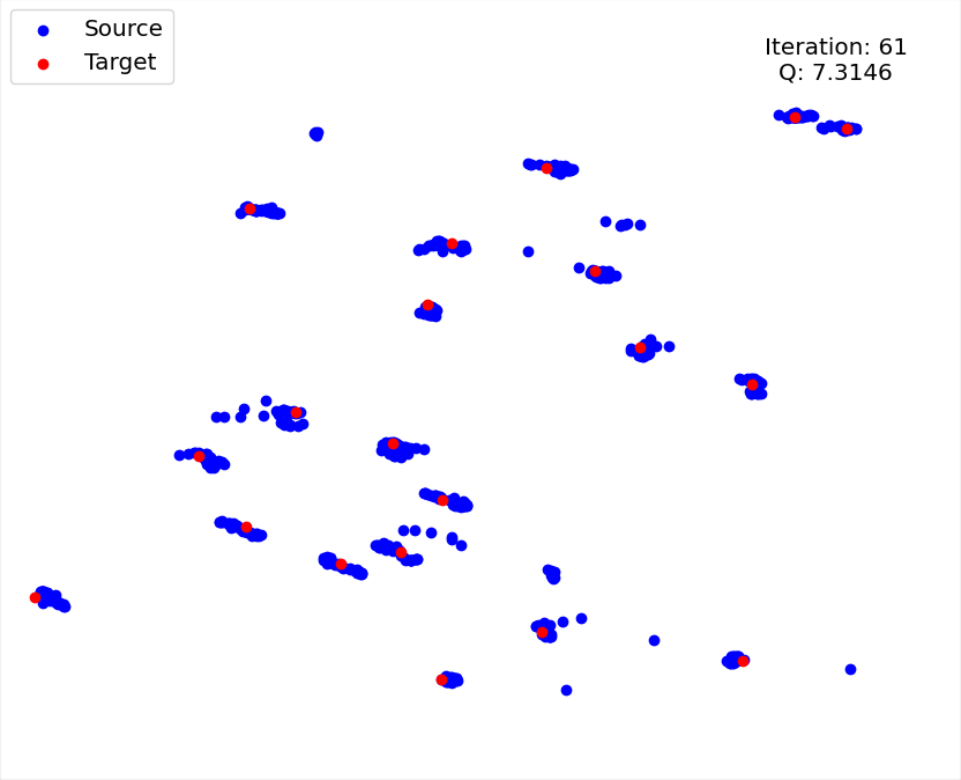


Figure 4.3: Result of Coherent Point Drift – matched unrolled plane of detections (blue) with the reference wall (red). The wall might not be recognizable, as this visualisation uses a transformed coordinate space.

4.5.1 Computing the transformations

As described in 2.4, the absolute transformations are computed for each frame using pairs of detection points and the corresponding reference wall hold.

4.5.2 Removing outlier transformations

Even with precise detections, some absolute transformations may not be determined correctly. Partially covered holds may create a large error – especially in frames with few holds. Also, some parts of the speed-climbing wall have holds placed on the same line, meaning that there is not enough information to construct the full transformation. These and similar situations lead to sub-optimal absolute transformations produced in the previous step.

Since it is expected that the transformations do not deviate much from the "no-op transformation" except for the translation element, it is possible to define a threshold for acceptable transformations as a difference from the unit matrix. All transformations not within this threshold are discarded, and the remaining ones are marked as keyframes.

4.5.3 Interpolating transformations

Since the previous step discards some transformations, this step attempts to interpolate transformations from the neighboring frames. This is done by utilizing the relative transformations computed in section 4.3.2. For each "gap" between keyframes, a forward transformation is computed for each frame in the gap by applying the relative transformations of the frames in the gap to the absolute transformation of the preceding keyframe. Conversely, a backward transformation is computed. An average of these two transformations, weighed by frame distances to previous and next keyframes, is then used as the interpolated absolute transformation of each frame in the gap. For gaps at the start and end of the video, only forward or backward transformation is used.

4.5.4 Smoothing transformations

Since the previous steps do not take temporal continuity into account, the computed absolute transformations often appear "shaky" and "noisy". Since camera movements are expected to be relatively smooth, the last step is to smoothen the absolute transformations using cubic spline interpolation.

However, there is one abrupt camera movement present in most videos – the start, where the camera stays still for a moment before starting to follow the climber. In order to not "smooth out" the start, a 100 times larger weight is given to the first 12 starting frames when computing the splines.

4.6 Summary

The pipeline uses an object detection model to detect hold positions. The detections are then tracked using optical flow, and registered to the reference wall using Coherent Point Drift.

The final result is a smooth sequence of transformation matrices, projecting each corresponding frame to the reference wall.

In chapter 5, more details are offered on the technology choices. Effectiveness of the pipeline is then evaluated in chapter 6.

5 Data and implementation

This chapter further elaborates on the previous chapter, discussing data manipulation in detail and specific technology choices. The complete pipeline code is attached to this thesis, as described in appendix A.

5.1 Dataset

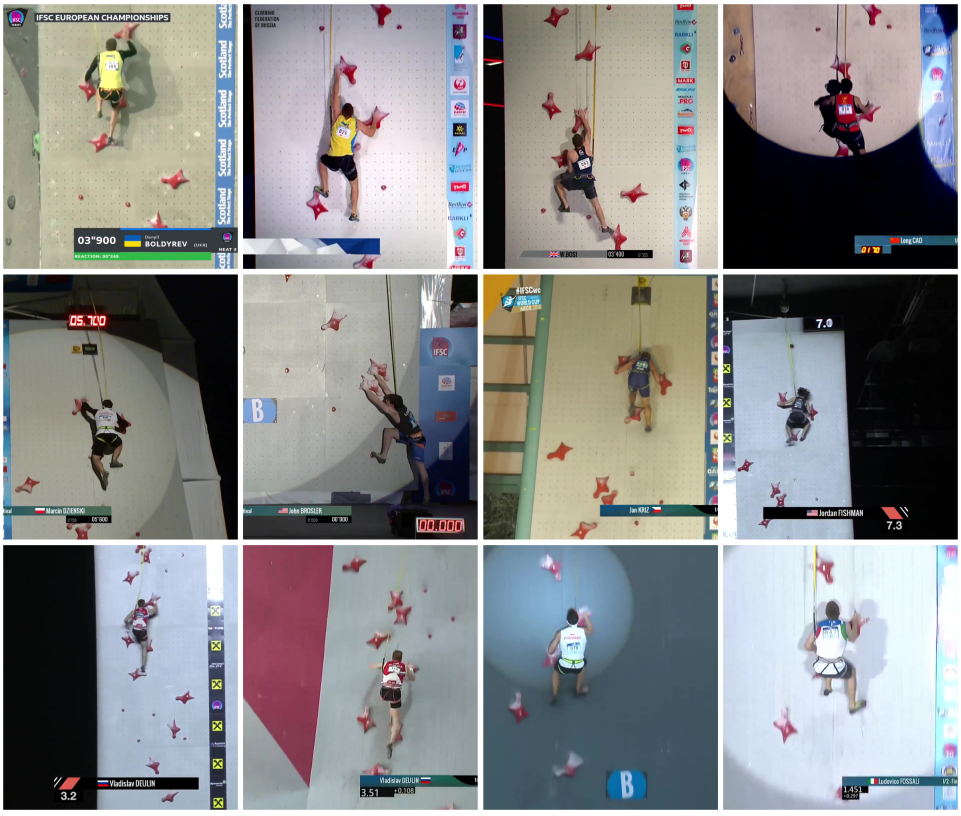


Figure 5.1: Example stills from the video dataset.

The dataset contains 367 videos in MPEG-4 format of speed climbing performances (see Figure 5.1). Veronika Škvarlová [21] extracted the videos from the recordings of speed climbing world finals pub-

lished by the International Federation of Sport Climbing on YouTube[1]. The beginnings and ends of the videos are aligned with the start and finish of the performance. The frame is cropped, so only one climber and the corresponding wall are shown in the frame.

The dataset videos are all normalized to the size of 960×1080 pixels.

The framerates differ: 80% of videos are in 25 frames per second, 12% in 30 frames per second, and 8% in 50 frames per second. Furthermore, some videos appear to be interpolated from a different framerate, which is possible given that the videos went through several steps of post-processing between being captured and being uploaded to YouTube.

The videos are shot across several competitions worldwide, occurring between the years 2018 and 2020. While the hold shape and color are consistent across competitions, the surroundings vary. Some videos capture parts of adjacent walls, showing holds that are not part of the analyzed wall. The light conditions vary, with some videos shot in darkness with just the climber illuminated by a stage light. Overlay graphics (showing information like the climber's name) are present in most videos. Some of the videos have significant compression artifacts. Overall, the dataset videos represent a wide range of real-world conditions.

5.2 Hold detection

5.2.1 Preparing training data

In order to train the hold detector, training data is needed. Using the video-processing command line tool `ffmpeg` [22], a frame is sampled from the dataset videos every 2 seconds, and 500 frames are hand-labeled using `labelImg` [23], a graphical program for labeling images with bounding boxes.

`labelImg` saves the labels in the Pascal Visual Object Challenge XML format [24]. In order to use them in the next step, a conversion to the `tfrecord` format native to TensorFlow is performed using a script created by Lyudmil Vladimirov [25].

1. `https://www.youtube.com/channel/UC2MGuhIaOP6YLpUx106kTQw`

5.2.2 Object detection framework

It was chosen to use the TensorFlow 2 [11] Object Detection API [26], specifically the model "SSD MobileNet V2 FPNLite 320x320" [20]. TensorFlow is a de-facto industry standard in machine learning, focusing on both research and practical applications (e.g., mobile inference). The associated model zoo[2] contains state-of-the-art object detection models, and the API offers a modern declarative way of defining training parameters for fine-tuning. These features allow for easy extendability, for example, replacing the model with a stronger one in the future.

5.2.3 Training the model

Training a deep learning model is a computationally intensive task, even when only fine-tuning an existing model, as is the case here. Commonly, specialized hardware is utilized for this task, like TPUs (Tensor Processing Units), which TensorFlow creators used to train the original model. However, since no specialized hardware was available, a consumer GPU (NVIDIA RTX 2070) was used for fine-tuning the model.

The dataset size of 500 images is relatively small. In order to prevent overfitting on the training set and improve generalization, we employ a set of data augmentations. The training script creates new variants of each training sample by randomized operations that alter the image but preserve the semantics[3]. These include flips and rotations, cropping and padding, color distortion (brightness, contrast, saturation), and JPEG compression.

The model is then left to train on for 20 thousand steps, taking several hours.

---

2. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`
3. For example, an image depicting a hold in a specific position will still depict it when 5% darker.

5.3 Pipeline implementation

5.3.1 Programming language

We decided to use Python 3 due to its extensive catalog of libraries, often covering cutting-edge research. Although Python is a high-level language, meaning that it offers lower performance than low-level languages, the data processing tasks mostly happen in compiled libraries offering native performance.

5.3.2 Model inference

As mentioned before, TensorFlow 2 [11] Object Detection API [26] was used to train the object detector model. The inference is then run on the exported model using TensorFlow 2 Python API. Unlike training, a GPU is not necessary for the inference – the performance of a standard consumer CPU is satisfactory.

5.3.3 Computer vision

For video processing tasks, the OpenCV [18] computer vision library was used. Uses of this library in the pipeline include loading and saving video, adding graphics to video, and computing optical flows and frame transformations.

5.3.4 Scientific libraries

Multi-dimensional array operations are performed using the `numpy` [27] library. As a Coherent Point Drift [14] implementation, `pycpd` [28] was used. For the linear sum assignment algorithm [16] implementation and more, `scipy` [29] was used. `scikit-learn` [30] was used for DBSCAN clustering. `csaps` [31] was used for cubic spline interpolation.

5.4 Packaging and portability

Projects involving machine learning and scientific libraries are often dependent on a particular operating system environment. Therefore, the project was equipped with a configuration file for Docker [32], a

containerization solution. The usage of Docker ensures that the project (including model training on the GPU) can be run effortlessly on any modern machine, regardless of the OS.

The code including the Docker configuration is attached to this thesis. See Appendix A for more details.

6 Evaluation and substitution study

In the chapter 4, a pipeline was described. In this chapter, we perform a substitution study on the pipeline. For each step of the pipeline, we evaluate alternative approaches to ensure that the pipeline from chapter 4, referred to as "selected pipeline", performs the best among all the possible approaches.

The setting of the selected pipeline are as follows:

- Training options: 20k steps, augmentations
- Detection threshold: 50%
- Hold tracking method: Optical flow association, removing tracks shorter than 3
- Relative transformation calculation: Using hold association, 2 degrees of freedom
- Absolute transformation calculation: Using 8 degrees of freedom
- Cleaning: Using $\epsilon = 0.3$
- Smoothing: Cubic spline smoothing, coefficient 0.0005

6.1 Performance

On a modern CPU (AMD Ryzen 5 3600), processing a single 6-second video takes around 15 seconds, with the most demanding part being hold detection. The performance could probably be further improved by batched inference and parallelization of some pipeline stages.

6.2 Methodology

6.2.1 Substitution study

A substitution study consists of replacing different parts of the pipeline to evaluate their contribution. As such, in each experiment, all the stages but one remain as in the selected pipeline. This method assumes

that the contribution of each pipeline step can be measured independently and may miss possible improvements achievable by changing more than one pipeline step at once. In other words, this method does not disprove the possibility of the selected pipeline being a local, not global, maximum. A more thorough examination is, however, not possible without an automated metric.

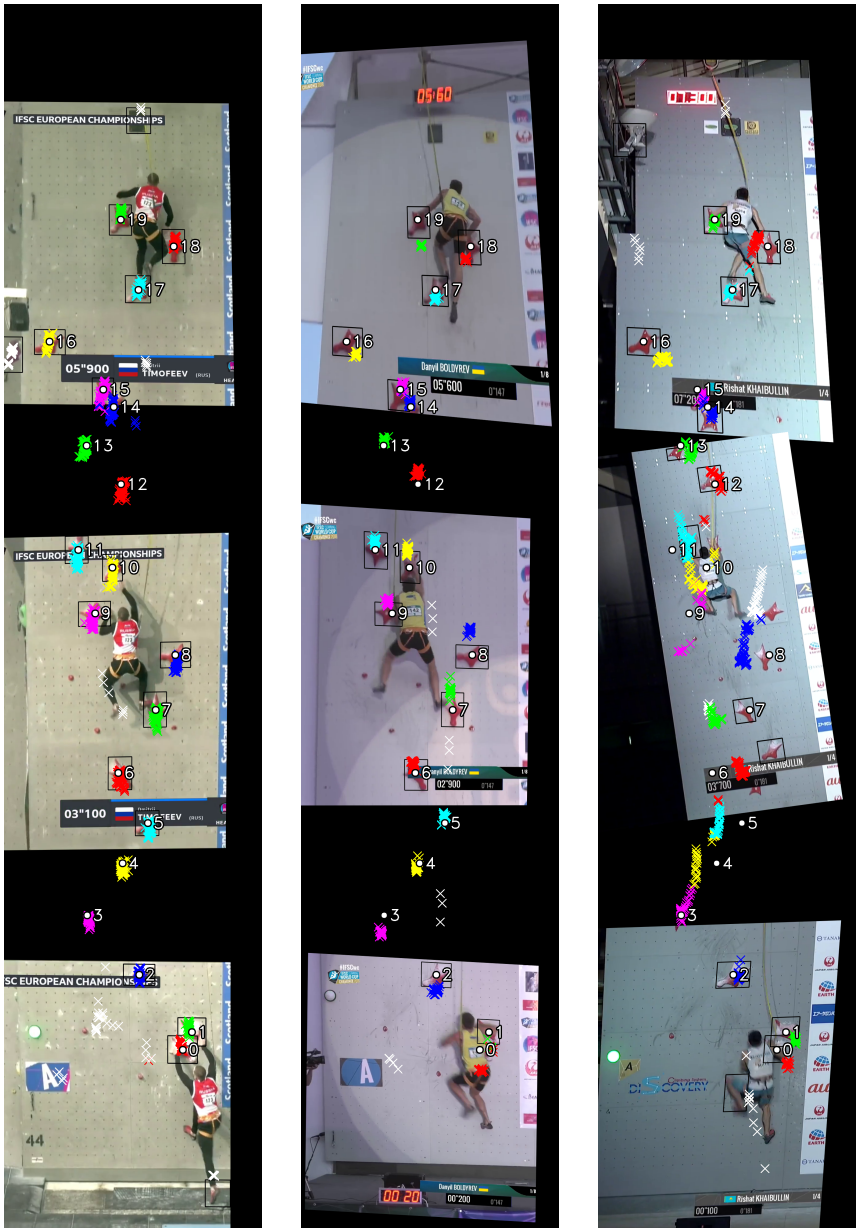
6.2.2 Chosen metric

Since there is no ground truth to compare against, it is impossible to construct an automated metric for evaluating the mapping quality. Therefore, we decided to create a scale of transformation quality that will be evaluated manually for each video.

From each source video, we generate an **evaluation video** by applying the computed series of transformations to video frames and overlaying with positions of the reference wall holds. Black rectangle outlines mark detected holds. The plane of unrolled detections as transformed by Coherent point drift is visualized using X's for detection positions, colored according to the matched reference hold: red, green, blue, magenta, yellow, cyan cyclically for matched ones, white for unmatched ones. We show examples of such evaluation videos in Figure 6.1.

In an ideal case, the holds visible in the video will align with the overlay, and the transformation will exactly copy the camera motion without any extraneous movement or deformations. We devised a rating scale as follows:

- level 5 – perfect mapping
- level 4 – small deviations, like short drift at the beginning or the end
- level 3 – small part of the video out of sync
- level 2 – large part of the video out of sync
- level 1 – video completely unusable



(a) good mapping (b) acceptable mapping (c) bad mapping

Figure 6.1: Examples of the evaluation visualisations, here presented as a combination of three frames from the evaluation video.

6.2.3 Evaluation dataset

The evaluation dataset was constructed by randomly choosing 20 videos from the entire dataset of 367 videos (as described in section 5.1). We skipped the videos already in the detector's training set to evaluate the detector on "fresh" data. We chose the number 20 since this amount of videos is possible to be scored by hand: with 24 experiments, 480 videos were scored.

6.2.4 Experiments

Experiments were split into categories. In each category, we compare the selected pipeline with alternative approaches. We generate a set of evaluation videos from the evaluation dataset for each experiment and assign scores by hand.

For each experiment category, we present a table where the first row (in bold) represents the selected pipeline, and other rows (sorted by average score, descending) represent other experiments. For each experiment, the average score and a histogram of scores are presented.

6.3 Hold detection

6.3.1 Model size

The model used for the pipeline is "SSD MobileNet V2 FPNLite", which with its size 320×320 input RGB pixels is one of the slimmest models in the TensorFlow 2 Object Detection API model ZOO[1]. It is quite probable that a larger model would perform better. However, the hardware for examining this possibility was not available, and evaluating the performance of a larger model remains a subject of further research.

1. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

6.3.2 Training

Our metrics

For each model published in the TensorFlow 2 Object Detection API's model ZOO, a default configuration is attached. Two types of training configurations were compared in this step – the original ones (with the only change of batch size from 128 down to 32 for hardware reasons) and our modified ones with added augmentations as outlined in section 4.2.

As seen in table 6.1, the combination of augmentations and training with 20 thousand steps turned out to be the most efficient. The model trained with 30 thousand training steps did most likely over-fit.

Standard object detection metrics

It should be noted that out of all stages of the pipeline, the object detection model has a numerical metric available: TensorFlow 2 Object Detection API supports model evaluation using the Average Precision and Average Recall metrics. However, this metric is designed for general multi-class object detection and thus is not entirely reflective of pipeline performance.

A set of 100 test images, taken from different videos than the training set, was used to evaluate the Average Precision and Average Recall metrics. As seen in Figure 6.1, these metrics favor the 20 thousand steps model without augmentations and the 30 thousand step model over the chosen model.

Training options	avg	#5	#4	#3	#2	#1	AP	AR@10d
20k steps, augmentations	**4.10**	**10**	**5**	**3**	**1**	**1**	**0.680**	**0.744**
30k steps, augmentations	3.95	11	2	3	3	1	0.687	0.759
20k steps, no augmentations	3.80	10	3	2	3	2	0.755	0.814
10k steps, augmentations	3.75	8	5	3	2	2	0.665	0.745

Table 6.1: Scores for different model training options, from our metric and from TF2 OD API metrics (Average Precision and Average Recall over 10 detections)

6.3.3 Detection threshold

Different detection thresholds, as outlined in section 4.2.3, were evaluated. It is expected that lower detection thresholds add some detections that might otherwise be missed, but at the same time add spurious detections. Thresholds of 30%, 50% (the selected pipeline), 70% and 90% were evaluated.

As seen in table 6.2, the threshold of 50% was found to be the most effective.

Detection threshold	avg	#5	#4	#3	#2	#1
Detection with threshold 50%	**4.10**	**10**	**5**	**3**	**1**	**1**
Detection with threshold 70%	3.95	9	5	3	2	1
Detection with threshold 90%	3.85	10	4	1	3	2
Detection with threshold 30%	3.85	7	6	5	1	1

Table 6.2: Scores for different detection thresholds.

6.4 Hold tracking

The selected method for hold tracking, as described in section 4.3.1, uses optical flow for associating the holds, and tracks of length lower

than 3 are removed. Different thresholds for track removal are evaluated $(0, 3, 6, 12)$. Higher values perform better in removing spurious detections while also increasing the possibility of removing a valid hold track. Especially the topmost holds are shown in the video for a very short time, meaning that too eager track removal could remove their tracks and cause a drift at the end of the video.

A variant where instead of using the position projected using optical flow, a previous position is used is also evaluated. Figure 4.2 shows the flow estimations as blue dots and previous positions as red dots. It is apparent that when the spacing of holds is large enough, this distinction should not make a difference. There might, however, be a difference when the holds are close together, especially in zoomed-out low-framerate videos.

As seen in table 6.3, threshold 6 has the same efficiency as threshold 3, while threshold 12 has slightly lower efficiency, probably due to the removal of "good" tracks. Association without short track removal leads to worse results, which can be explained by spurious detections continuing into the following pipeline steps.

Previous position association has a slightly lower score than optical flow association, most likely due to some holds being tracked incorrectly.

Hold tracking method	avg	#5	#4	#3	#2	#1
Optical flow association, removing tracks shorter than 3	**4.10**	**10**	**5**	**3**	**1**	**1**
Optical flow association, removing tracks shorter than 6	4.10	10	5	3	1	1
Optical flow association, removing tracks shorter than 12	4.05	10	3	6	0	1
Previous position association, removing tracks shorter than 3	4.05	9	6	3	1	1
Optical flow association, no short track removal	4.00	9	6	2	2	1

Table 6.3: Scores for different hold tracking methods

6.5 Calculating relative transformations

In the selected pipeline, hold associations are used to compute the relative transformations between two frames in 2 degrees of freedom (i.e., translation only). For every two consecutive frames, pairs of old and new tracked holds are used to compute the transformation.

A variant where 4 degrees of freedom is used instead (allowing uniform scaling and rotation) is evaluated.

In an alternative approach, a method where optical flow is used to estimate the transformation is used. A standard computer vision solution for this task is extracting features[2] from the source frame and calculating optical flow to the destination frame. However, this approach is not suitable for the class of videos this thesis deals with since most of the scene's features come from a dynamic object – the climber, as shown in Figure 6.2. It is thus necessary to explicitly track the wall and not the climber. Detection boxes are used to mask holds (which move in the direction we are interested in), corners are extracted using the "Good Features to Track" algorithm [33]. Their optical flow vector is then computed using a Lucas-Kanade feature tracker [17]. Vectors that do not land in one of the detection boxes in the next frame are discarded. An average value of all the flow vectors is then used as a translation estimation.

As seen from the results in 6.4, using the alternative optical-flow-based approach is much less precise than using hold association, with most videos from score 5 moving to score 4.

Hold association with 4 degrees of freedom performs terribly, with no videos with a score of more than 3. The accumulation of errors in the detection plane unrolling (see section 4.4.1) is most likely too high to produce a feasible plane.

2. A set of points that are readily identifiable between frames, like sharp corners.

Figure 6.2: Red dots represent features detected by a library function from OpenCV, most of which are located on the climber and stationary graphics. This illustrates how a targeted approach, using hold detections, is needed in order to estimate relative transformations correctly.

Relative transformation calculation	avg	#5	#4	#3	#2	#1
Using hold association, 2 degrees of freedom	**4.10**	**10**	**5**	**3**	**1**	**1**
Using optical flow, 2 degrees of freedom	3.80	5	10	2	2	1
Using hold association, 4 degrees of freedom	1.70	0	0	3	8	9

Table 6.4: Scores for different relative transformation calculation methods.

6.6 Calculating absolute transformations

6.6.1 Degrees of freedom

Absolute transformations are computed using the assignment of detection points to the reference wall points. In the selected pipeline, the assignment is performed using a combination of unrolling the detections onto a common plane using relative transformations from the previous step, finding a transformation aligning it with the reference plane using Coherent point drift [14], and finally using linear assignment with a threshold to associate points in each frame.

Different degrees of freedom for the absolute transformation computation are evaluated. (See section 3.4 for more information on degrees of freedom in transformation matrices.) While the transformation should naturally use the full 8 degrees of freedom, since a perspective transformation is present in many videos of the dataset, it is beneficial to examine the possibility of using fewer degrees of freedom as a trade-off of perspective accuracy for increased precision in other degrees.

6.6.2 Clustering

An alternative approach employs a clustering algorithm before running Coherent point drift. Each cluster is then represented by its centroid.

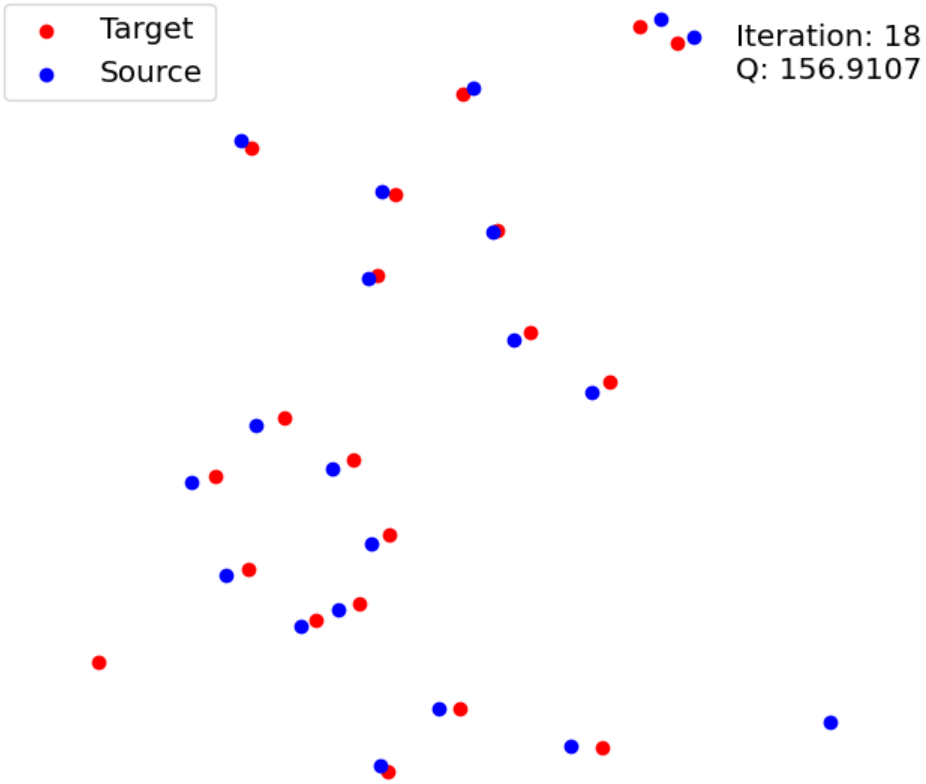


Figure 6.3: Result of the Coherent point drift algorithm over detections clustered by DBSCAN. Red dots represent the reference wall, blue holds the detection cluster centroids. A false negative (unmatched reference hold) can be observed in the bottom left, and a false positive (unmatched detection cluster centroid) in the bottom right. Compare with Figure 4.3 that shows the non-clustered approach.

It is not possible to use standard K-means clustering since K is not known in advance – the expected number of holds is known, but false positives may create spurious clusters.

It was decided to use the clustering algorithm DBSCAN [34] available from `scikit-learn` [30]. It has the advantage of detecting the correct number of clusters and being resilient to clusters having stretched or non-convex shapes – something that can happen due to the way the plane unrolling is performed. Another positive property of this clustering algorithm is that not all input points need to be a part of a cluster – points can be outliers, like spurious detections.

6.6.3 A naïve method assuming precise detections

It is possible to connect "severed" tracks of the same physical hold by estimating the hold's position once the track disappears. This approach enables to associate tracks with reference holds directly, without employing further steps in the pipeline. Very precise detections are required for this approach to work, which is not a reasonable assumption – some videos in the dataset even crop out some holds fully, never showing them on screen. However, while naïve, this approach is surprisingly functional for "good" videos and worth mentioning.

Since some detection tracks may be severed by an occasional false negative from the object detector or simply by covering the hold with the climber's body, one physical hold will usually result in multiple detection tracks. In order to partially mitigate this, a "fake" detection is inserted and marked as such to the projected position whenever there is no actual detection to associate it with, as shown in Figure 6.4. "Fake" detections continue to be tracked until they leave the frame or associate with an actual detection in some of the following frames.

"Fake" detections are not used for further steps when a precise detection point position is required – their purpose is only to connect severed tracks.

The tracks are then associated from bottom to top to corresponding reference wall holds in order of appearance. When the input video is detected correctly (no spurious detections left after removing short tracks, all tracks correctly connected), this results in the correct detection registration.

Figure 6.4: Full circles represent actual detections, empty circles represent "fake" detections. Arrows show the position of the associated detection in the next frame.

This method was not evaluated in the substitution study since it does not integrate well with the rest of the pipeline. However, it is included here for completeness, as it could be helpful in further research if a better hold detector is developed.

6.6.4 Results

As seen in table 6.5, the best results are achieved using full 8 degrees of freedom, with 6 and 4 degrees of freedom showing worse mapping quality. It is to be noted, though, that depending on the use case of the obtained absolute transformations, it might still be possible to prefer fewer degrees of freedom (which is easily configurable in the pipeline implementation).

The variant with DBSCAN clustering scores low, showing that pre-clustering holds separately leads to a loss of information.

Absolute transformation calculation	avg	#5	#4	#3	#2	#1
8 degrees of freedom	**4.10**	**10**	**5**	**3**	**1**	**1**
6 degrees of freedom	3.95	10	2	5	3	0
4 degrees of freedom	3.95	8	4	7	1	0
8 degrees of freedom, clustered	3.65	7	5	4	2	2

Table 6.5: Scores for different absolute transformation calculation methods.

6.7 Cleaning absolute transformations

The series of absolute transformations is cleaned by removing outliers, which are detected by an element-wise difference from the unit matrix (except for the translation vector). The maximum difference from unit matrix permittable is labeled ϵ. While small values of ϵ may cause correct transformations to be dropped, too high values may introduce some outliers to further steps, causing erratic movement.

Values 0.2, 0.3 (as in selected pipeline) and 0.4 were evaluated for ϵ. As shown in the table 6.6, best results were achieved with $\epsilon = 0.2$.

Cleaning absolute transformations	avg	#5	#4	#3	#2	#1
Cleaning with $\epsilon = 0.3$	**4.10**	**10**	**5**	**3**	**1**	**1**
Cleaning with $\epsilon = 0.4$	4.05	7	10	1	1	1
Cleaning with $\epsilon = 0.2$	3.85	9	4	4	1	2

Table 6.6: Scores for different cleaning constants

6.8 Smoothing absolute transformations

Absolute transformations output by the previous step are calculated separately for each frame. Since the absolute transformations are supposed to correspond to camera movement, a smoothing algorithm is employed.

Smoothing coefficients 0.005, 0.0005, and 0.00005 are evaluated, and variants of the selected pipeline's smoothing coefficient (0.0005) with the weight of non-keyframes adjusted to 0 and 0.5, as well as the increased weight for starting frames disabled.

As an alternative approach, a cubic Savitzky-Golay filter [35] with window length 49 is used instead of the cubic spline smoothing. A Savitzky-Golay filter operates with each datapoint separately, fitting a polynomial to a window centered around the datapoint and then smoothing the value by evaluating the polynomial. This approach is different from spline smoothing, which operates on the whole sequence and fits the splines between well-defined "knots".

As seen from the results in table 6.7, the smoothing coefficients 0.0005 and 0.00005 offer equivalent performance, with 0.005 not being "smooth enough". No increased starting weight helps a small number of videos reduce the initial drift. Lowering the weight for non-keyframes lowers the score by a large margin. Finally, the Savitzky-Golay filter outputs a smooth result, but the nature of the algorithm does not smooth out outliers as correctly as the cubic spline smoothing.

Smoothing method	avg	#5	#4	#3	#2	#1
Cubic spline smoothing, coefficient 0.0005	**4.10**	**10**	**5**	**3**	**1**	**1**
Cubic spline smoothing, coefficient 0.00005	4.10	10	5	3	1	1
Cubic spline smoothing, coefficient 0.0005, no increased start weight	4.05	9	5	4	2	0
Cubic spline smoothing, coefficient 0.005	3.95	8	6	4	1	1
Cubic spline smoothing, coefficient 0.0005, keyframes have double weight	3.90	6	9	3	1	1
Savitzky-Golay filter, window size 49	3.75	4	11	2	2	1
Cubic spline smoothing, coefficient 0.0005, only keyframes	3.40	3	7	6	3	1

Table 6.7: Scores for different absolute transformation smoothing options.

6.9 Study conclusion

From the evaluation dataset of 20 videos, the chosen pipeline output achieved the highest score on 10 videos, with 5 more videos scoring one less. The substitution study showed the influence of different constants and approaches in all steps of the pipeline.

Common mapping errors included drift at the start and at the end of a video, where not enough holds are visible to find a good transformation, and "shakiness" of the transformation sequence, caused by imprecise hold detections.

It was shown that the method of computing relative transformations has a significant influence on the overall success of the pipeline, thus pinpointing this area as a candidate for future research.

The most notable omission from this substitution study was evaluating larger object detection models, which was skipped due to hardware reasons. However, we expect larger models to perform better, thus sending higher-quality detections to further pipeline steps.

7 Conclusion

A pipeline for finding transformations between speed climbing videos and a reference wall was designed, implemented, and evaluated. The main steps of the pipeline are hold detection using an object detection model trained on custom data, detection tracking using optical flow, and hold registration in which the detections are unrolled on a common plane and then mapped to the reference wall holds using the Coherent point drift algorithm.

We have implemented the pipeline using Python 3, TensorFlow 2 Object Detection API, OpenCV, and several more scientific libraries. In addition, Docker is used as a containerization solution to ensure better compatibility across platforms. The complete pipeline source code is attached (see Appendix A).

In the evaluation chapter, we proposed a manual metric to determine a mapping effectiveness score for a video. We have then shown that the selected pipeline design maximizes this metric. We observed that the method of obtaining relative transformations between frames has a significant influence on the overall result, pinpointing this stage as a subject for further research. The lack of ground truth data hindered the effectiveness of the evaluation. To further improve the pipeline in further research, a dataset with ground truth associated must be obtained.

The plan for future research is to analyze and compare nuances in top-level athletes' speed climbing styles. For this analysis the pipeline proposed in this work will be employed to obtain the unrolled skeleton data from hundreds of climbs.

A Attachments

The attachments to this thesis are as follows:

`speed-climbing-mapping.zip` is an archive containing source code for the pipeline. It includes:

- The full source code of the pipeline
- Hold detection training dataset
- Pre-trained weights for all 4 evaluated models
- Docker configuration, allowing easy setup on a new machine
- Evaluation dataset of 20 videos (in the `test_data` subfolder)
- Evaluation videos generated by the selected pipeline (in the `out` subfolder)

For more details and usage, see the included `README.md` file.

The code is also published to GitHub[1]. The published version does not include the data and pre-trained models.

`substitution-study.xlsx` is a Microsoft Excel file containing source data for the substitution study performed.

1. `https://github.com/JanPokorny/speed-climbing-mapping`

Bibliography

1. PAVLLO, D.; FEICHTENHOFER, C.; GRANGIER, D.; AULI, M. *3D Human Pose Estimation in Video with Temporal Convolutions and Semi-Supervised Training* [online]. 2019-03-29 [visited on 2021-05-14]. Available from arXiv: 1811.11742 [cs].
2. BOCHKOVSKIY, A.; WANG, C.; LIAO, Hong-Yuan Mark. *YOLOv4: Optimal Speed and Accuracy of Object Detection* [online]. 2020-04-22 [visited on 2021-05-14]. Available from arXiv: 2004.10934 [cs, eess].
3. LEGNER, S. *Climbing World Championships 2018 Speed* [online]. 2018 [visited on 2021-05-01]. Available from: https://commons.wikimedia.org/wiki/File:Climbing_World_Championships_2018_Speed_(DSC09055).jpg.
4. PUEO, B.; JIMENEZ-OLMEDO, J. Application of Motion Capture Technology for Sport Performance Analysis. *Retos: nuevas tendencias en educación física, deporte y recreación*. 2017, vol. 2017, pp. 241–247.
5. REVERET, L.; CHAPELLE, S.; QUAINE, F.; LEGRENEUR, P. 3D Visualization of Body Motion in Speed Climbing. *Frontiers in Psychology* [online]. 2020, vol. 11 [visited on 2021-05-14]. ISSN 1664-1078. Available from DOI: 10.3389/fpsyg.2020.02188.
6. LEGRENEUR, P.; ROGOWSKI, I.; DURIF, T. Kinematic Analysis of the Speed Climbing Event at the 2018 Youth Olympic Games. *Computer Methods in Biomechanics and Biomedical Engineering* [online]. 2019, vol. 22, S264–S266 [visited on 2021-05-11]. ISSN 1025-5842, ISSN 1476-8259. Available from DOI: 10.1080/10255842.2020.1714907.
7. IGUMA, H.; KAWAMURA, A.; KURAZUME, R. A New 3D Motion and Force Measurement System for Sport Climbing. In: *2020 IEEE/SICE International Symposium on System Integration (SII)*. 2020, pp. 1002–1007. ISSN 2474-2325. Available from DOI: 10.1109/SII46433.2020.9026213.

8. CAO, Z.; HIDALGO, G.; SIMON, T.; WEI, S.; SHEIKH, Y. *OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields* [online]. 2019-05-30 [visited on 2021-05-14]. Available from arXiv: 1812.08008 [cs].

9. LIN, T.; GOYAL, P.; GIRSHICK, R.; HE, K.; DOLLÁR, P. *Focal Loss for Dense Object Detection* [online]. 2018-02-07 [visited on 2021-05-14]. Available from arXiv: 1708.02002 [cs].

10. LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S., et al. *SSD: Single Shot MultiBox Detector* [online]. 2016 [visited on 2021-05-14]. Available from DOI: 10.1007/978-3-319-46448-0_2.

11. ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z., et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems* [online]. 2016-03-16 [visited on 2021-05-01]. Available from arXiv: 1603.04467 [cs].

12. GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge, Massachusetts: The MIT Press, 2016. Adaptive Computation and Machine Learning. ISBN 978-0-262-03561-3.

13. DERPANIS, Konstantinos G. Overview of the RANSAC Algorithm. *Image Rochester NY*. 2010, vol. 4, no. 1, pp. 2–3.

14. MYRONENKO, A.; SONG, X. Point-Set Registration: Coherent Point Drift. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2010, vol. 32, no. 12, pp. 2262–2275 [visited on 2021-04-17]. ISSN 0162-8828. Available from DOI: 10.1109/TPAMI.2010.46.

15. MOUNT, David M; NETANYAHU, Nathan S; MOIGNE, Jacqueline Le. Efficient Algorithms for Robust Feature Matching. *Pattern Recognition* [online]. 1999, vol. 32, no. 1, pp. 17–38 [visited on 2021-05-04]. ISSN 00313203. Available from DOI: 10.1016/S0031-3203(98)00086-7.

16. CROUSE, David F. On Implementing 2D Rectangular Assignment Algorithms. *IEEE Transactions on Aerospace and Electronic Systems* [online]. 2016, vol. 52, no. 4, pp. 1679–1696 [visited on 2021-05-03]. ISSN 0018-9251. Available from DOI: 10.1109/TAES.2016.140952.

17. BOUGUET, J. Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker. *Intel corporation*. 1999, vol. 5, no. 1-10, p. 4.

18. BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 2000, vol. 25, pp. 120–125.

19. IFSC SPORT DEPARTMENT. *Speed Licence Rules* [online]. IFSC, 2014 [visited on 2020-11-29]. Available from: `https://cdn.ifsc-climbing.org/images/ifsc/Footer/Manufacturers/140429_SDSpeedLicenseRules41-corrected.pdf`.

20. SANDLER, M.; HOWARD, A.; ZHU, M.; ZHMOGINOV, A.; CHEN, L. *MobileNetV2: Inverted Residuals and Linear Bottlenecks* [online]. 2019-03-21 [visited on 2021-03-14]. Available from arXiv: `1801.04381 [cs]`.

21. ŠKVARLOVÁ, V. *Labeled Dataset of Speed Climbing Performances*. Brno, 2021. Available also from: `https://is.muni.cz/th/ll5rp/`. Bechelor's thesis. Masaryk University.

22. TOMAR, S. Converting Video Formats with FFmpeg. *Linux Journal*. 2006, vol. 2006, no. 146, p. 10.

23. LIN, Tzu Ta. *Tzutalin/labelImg* [online]. 2015 [visited on 2021-05-03]. Available from: `https://github.com/tzutalin/labelImg`.

24. EVERINGHAM, M.; ESLAMI, S. M. A.; VAN GOOL, L.; WILLIAMS, C. K. I.; WINN, J.; ZISSERMAN, A. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision* [online]. 2015, vol. 111, no. 1, pp. 98–136 [visited on 2021-05-01]. ISSN 0920-5691, ISSN 1573-1405. Available from DOI: `10.1007/s11263-014-0733-5`.

25. VLADIMIROV, L. *Training Custom Object Detector — TensorFlow 2 Object Detection API Tutorial Documentation* [online]. 2020 [visited on 2021-05-01]. Available from: `https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html`.

26. HUANG, J.; RATHOD, V.; SUN, C.; ZHU, M.; KORATTIKARA, A., et al. *Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors* [online]. 2017-04-24. Version 3 [visited on 2021-04-27]. Available from arXiv: `1611.10012 [cs]`.

27. HARRIS, C. R.; MILLMAN, K. J.; van der WALT, S. J.; GOMMERS, R.; VIRTANEN, P., et al. Array Programming with NumPy. *Nature*. 2020, vol. 585, no. 7825, pp. 357–362. Available from DOI: 10.1038/s41586-020-2649-2.

28. KHALLAGHI, S. *Siavashk/Pycpd* [online]. 2021 [visited on 2021-05-03]. Available from: https://github.com/siavashk/pycpd.

29. VIRTANEN, P.; GOMMERS, R.; OLIPHANT, Travis E.; HABERLAND, M.; REDDY, T., et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* [online]. 2020, vol. 17, no. 3, pp. 261–272 [visited on 2021-05-06]. ISSN 1548-7091, ISSN 1548-7105. Available from DOI: 10.1038/s41592-019-0686-2.

30. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B., et al. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830.

31. PRILEPIN, E. *Espdev/Csaps* [online]. 2021 [visited on 2021-05-03]. Available from: https://github.com/espdev/csaps.

32. MERKEL, D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014, vol. 2014, no. 239. ISSN 1075-3583. Available from DOI: 10.5555/2600239.2600241.

33. SHI, J.; TOMASI. Good Features to Track. In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. ISSN 1063-6919. Available from DOI: 10.1109/CVPR.1994.323794.

34. SCHUBERT, E.; SANDER, J.; ESTER, M.; KRIEGEL, Hans Peter; XU, X. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Transactions on Database Systems* [online]. 2017, vol. 42, no. 3, pp. 1–21 [visited on 2021-04-28]. ISSN 0362-5915, ISSN 1557-4644. Available from DOI: 10.1145/3068335.

35. SAVITZKY, Abraham.; GOLAY, M. J. E. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry* [online]. 1964, vol. 36, no. 8, pp. 1627–1639 [visited on 2021-05-06]. ISSN 0003-2700. Available from DOI: 10.1021/ac60214a047.