UiO : Department of Informatics
University of Oslo

Human Action Retrieval in the sport analytic system BAGADUS

Inferring Action Labels and 2D Skeletons from Video Sequences

Bård Eirik Winther

Master's Thesis Autumn 2015

Human Action Retrieval in the sport analytic system BAGADUS

Bård Eirik Winther

8th September 2015

Abstract

Sport analytic systems have had an increasing interest in the past few years, with advancements automatizing otherwise tedious and time consuming tasks. However, these systems are still not able to perform semantic analysis of video footage, such as in automatic activity labeling (for example "run" and "walk") and performance analysis (for example reaction times). In this thesis, we propose a prototype, within the Bagadus sport analytic system, allowing to obtain action labels and athelete poses. With these, it is possible to automatically annotate (label) activities and aid in analyzing the athletes performances (e.g., identify wether they use the right technique when running) as the athelete pose will be known.

Our prototype solves both the action recognition and pose estimation problems as a content-based video retrieval problem, that is, we first obtain athlete-centered video sequences and then compare these sequences against an annotated database. The comparison utilizes a video similarity measure, based on the motion occurring in the video sequences using optical flow. This approach allows us to obtain additional semantic information, consisting of action labels and poses and is suitable to the use-case of a soccer stadium where cameras are located at a distance from the atheletes. To support the proposed content-based video retrieval solution, a large set of athelete-skeletons (placement of joint lications for an athelete within a frame) is annotated using the crowdsourcing platform Microworkers. Using crowdworkers, we obtain high qualtiy skeletons that are comparable to expert annotations within a short period of time, which is achieved by iterating over several designs on the user interface and utilizing different filtering techinques on the annotations.

The proposed method presents a good performance in terms of accuracy and robustness for both action recognition and pose estimation, which correcly classifies 78% of all the actions for a set of selected video sequences and estimates poses with up to pixel-perfect results. This allows us to extend the sport analytic system Bagadus capabilities by including semantic analysis of actions and poses, but can also be used in entertainment applications such as free-view rendering.

Contents

List of Figures

List of Tables

List of Code Snippets

Chapter 1

Introduction

1.1 Background

There has been an increased interest in analyzing sports using video and sensors for tracking the atheletes [52] and annotating a match with *key events* (for example every time an athelete scores a goal in a competition). To better support and maintain large quantities of data and different recording systems, sport analytic systems are created to automatize and simplify the process of event logging and data management. For example, using video provides showcases of performances, often right after a competition. With such a system, a coach can faster and more precisely examine previous competitions and improve the skills of the atheletes.

Bagadus [49] is a sport analytic system designed to automate common tasks performed manually and experiment with new possibilities. With a camera array, ZXY player tracking and manual annotating of key events, Bagadus provides a large amount of data that can be used for any purpose. For example, this data can quickly be retrieved for analyzing competitions in great detail or for entertainment purposes. Additionally, Bagadus provides a panorama view, stitched from an array of cameras, with the possibility of having a virtual camera moving within this panorama view. However, this virtual view is limited to the single position of the camera array.

What is missing in the Bagadus system is automatic semantic analysis of the athelet's performance, e.g., automatic key events logging and assert if the movement technique is optimal. Having semantic information about actions and poses can be used to obtain a more detailed picture of a game which can then be used to tailor the training needed for the atheletes. Semantic analysis is currently done manually or limited to the data available from the ZXY player tracking system (position data on the field). To automate this process, we require information of the action performed (e.g., "run" or "kick") and the pose of the athelete (i.e., a skeleton with joint positions that defines an atheletes pose). A technique based on analysing video sequences and not on Motion Capture is therefore neccessary.

A direct solution to obtain actions and athelete poses is to use systems that attaches accelerometers or markers on the actor's individual limbs and is then recorded by specialized hardware, e.g., Motion Capture. *Motion Capture* is used extensively in film and video games, as done in [36], where the actor is placed with markers that are recorded synchronous with several cameras covering different angles. However, this technique is not possible to use with Bagadus, because it does not have a motion capture system. Installing it is not an option either, because placing motion capture markers is far too intrusive to be considered as an option.

1.2 Problem Statement

In this thesis, we aim to add semantic analyses in Bagadus by researching the two problems of automatic action recognition and pose estimation from video captured by the Bagadus system. Specifically, we want to have a complete game annotated with action labels and poses, with the poses represented by skeletons, for every single participating player.

1.2.1 Terminology

The problem solved in this thesis is first to infer semantic information from video sequences. Specifically, identify the action performed and secondly to obtain the corresponding pose for each frame for a given person. From the taxonomy presented by Moeslund et al. [34], we are interested in performing *action recognition* and *pose estimation* on an *actor*.

An actor is a person to whom the action or pose is to be obtained from, that in our context is a soccer player. A player performs *actions*, where a single action contains a sequence of *atomic actions*, which is then *recognized* by a program that usually provides an action label (e.g., "walking") as output.

The *atomic actions* consists of a single joint rotation or movement and is the smallest individual operation an actor can perform. For example, the action "throwing a ball" consists of atomic actions that makes up the individual arm movements in order to propel the ball forwards. Likewise, a set of actions can form an *activity*. For example, the actions kicking, running and standing are used to perform the activity of playing soccer.

Pose estimation is strongly related to action recognition as its goal is to obtain the individual atomic actions and model them as a *skeleton*. A *skeleton* is a model that describes the pose of an actor at a specific video frame and is commonly created as a graph. The graph model then uses nodes to represent joints and edges to represent limbs.

1.2.2 Specific Problem

The two main problems addressed in this thesis are to:

1. Infer semantic information about which actions the players are performing, e.g., "run" or "side-jump", based on a video sequence.

2. Estimate a model that describes the human body-pose for a particular frame, or in other words, estimate a two-dimensional skeleton of where joints and limbs are located within a video frame.

Since approaches like Motion Capture are not available, a solution using computer-vision techniques is therefore necessary and it must be able to infer the actions and poses based primarily from recorded video sequences. In essence, this involves isolating players on the field and analyse their movements so that their actions can be determined and their poses can be estimated, with video frames as the primary source of information. Using this techinque though, comes with certain limitations.

1.2.3 Assumptions and Limitations

The assumptions for this thesis originates from both the Bagadus hardware limitations and the scene of a soccer game. The hardware limitations typically comes from the quality and the accuracy of the camera sensors and ZXY tracker system, including video resolution. Additional limitations from the soccer game scenario includes the action types and player consistency in movement and appearance.

Assumptions

- A significant amount of noise is present in the video, consisting of motion blur and acquisition noise.

- A lot of video data is available, with redundant and repetitive player actions.

- The ZXY player tracking system has more data-entries than position, but these data-entries are not usable because they are incorrect and inconsistent.

- Bagadus can operate with other sports and events than soccer games and because of this, the solution must be adaptable to other locations and activities.

- Crowdworkers produce a varying degree of quality (including high and low quality work) for the answers they provide for tasks.

Limitations

- Players occupy only a small number of pixels, ranging from around 50 to 150 pixels in height, caused by camera-to-field distance and video resolution.

- Videos are aquired from a single viewpoint (i.e., one camera array).

- Only video sequences without occlusion and frequently occurring actions are used.

- Bagadus runs on commodity hardware and any new features or extensions should do this as well.

- We only use a small portion of a full soccer game to implement and test our proposed solution, as this makes the amount of data obtained and used manageable.

1.2.4 Approach

On the basis of the assumptions and limitations presented above, our proposed solution solves the two problems of inferring actions and estimating poses by shifting them to a single problem, which is the one of *content-based video retrieval* [46]. We do this because it allows us to solve two potentially challenging problems as a matching problem and significantly reducing the overall algorithmic complexity. Basically, content-based video retrieval systems fetch content similar to a query sequence based on content represented as a set of features such as shape, color or texture or other properties. In this context, the similarity measure is a matching function that returns the degree of similarity, i.e., how close two sequences resembles one another. An advantage of using content-based video retrieval is that it can be automatized to automatically encode video sequences as *feature vectors*, a vector containing key elements to describe properties of video sequence or frame, and query the information from a database. Video queries to a database returns similar sequences along with their semantic information without requirering additional metadata information (e.g., keywords) involving manual annotation.

Content-based video retrieval consists of two parts, namely an extraction algorithm to obtain feature vectors and an a matching functions for similarity measurement. The video retrieval system used in this thesis is the one proposed by Efros et al. [15]: It uses optical flow for the feature vectors which is used to compare the similarity of two sequences. Moreover, in [15], the authors solved the action recognition problem under very similar assumptions as ours, i.e., a soccer game with low resolution, high noise levels and a lot of motion blur in the video input. Furthermore, they describe a pose estimation technique that utilizes an annotated database of video sequences, along with 2D and 3D skeletons for different soccer player actions, in order to either synthesize or replicate action sequences. We do not have access to this dataset and must therefore create our own.

In short, solving the action recognition and pose estimation problems as a content-based video retrieval problem based on the solution by Efros et al. [15] consists in solving the three following procedures:

1. **Player Tracking**: Also referred to as *video extraction*, the goal of this step is to obtain player-centered sequences of soccer players by tracking and isolating individual persons from captured video. The resulting sequences are then used to build up the annotated database.

2. **Video annotation**: The process of annotating a database of player-centered sequences in order to obtain action labels and joint positions for every frame.

3. **Action retrieval**: Apply the content-based video retrieval system to obtain skeletons and action labels for query sequences from the annotated database, where the most similar database sequence's skeleton is reprojected onto the query sequence.

These three steps are the main topics covered in this thesis and represents the proposed solution, with each procedure as a program that together forms the following workflow:

1. Design and implement a robust player tracking algorithm that produces size-normalized, player-centered video sequences from data captured by the Bagadus system.

2. Develop a system allowing to distribute the annotation of action labels and skeletons to hundreds of people through a crowdsourcing platform. This is an uncommon task in the crowdsourcing community and we provide additional algorithms for quality assessments and task management.

3. Implement a parallel matching function for similarity comparisons and a skeleton transfer algorithm, both based on the solution proposed by Efros et al. [15].

4. Improve our implementation of the solution proposed by Efros et al. [15] by porting the classification algorithm to CUDA, nVidias GPU computing platform and Application Programming Interface (API).

1.3 Contributions

The work performed in this thesis has resulted in a real-world running prototype of an action recognition and pose estimation prototype. In short, the main contributions are:

1. A prototype for action recognition and pose estimation, which includes:
    - A complete solution consisting of everything from video extraction to rendered skeletons. Essentially, our proposed solution is complete and does not depend on other components to operate.
    - Robust player tracking and centering with automatic recovery and initialization: We obtain bounding boxes around players that are perfectly matched to the player's shape with the player perfectly in center.
    - Integration with the Bagadus system existing recording solutions and data formats. This allows it to be moved along the Bagadus system if it where to ever change or installed at a new location without having to do additional setup.
    - An annotation tool that is easy to use, reliable and secure. This tool is used for both experts and crowdworkers and is customizable depending on the annotation need.
    - Obtained data can be used directly from our programs or exported to other programs for external use. While the design is complete integration with the Bagadus system, we also provide support for external programs if desired.
2. We have run numerous experiments and validated the results, which includes:
    - Correctly classify 78% of all actions. This includes all types of actions, i.e., both ambigious (e.g., "kick") and simple, cyclic actions (e.g., "run"), as well as being able to distinguish between "walk", "run" and "sprint".
    - Obtain up to pixel-perfect estimation of poses. A 13-joint skeleton is obtained for every single frame in a sequence and can be achived as long as the query sequence's action class is present in the annotated database.
    - Optimizations for the similarity measure which achieves a speedup of almost nine times compared to the CPU. The speedup is compared to a parallel CPU implementation using OpenMP.
    - Filtering of crowdsourced annotated data to obtain results comparable with experts' annotations, despite being an unusual scenario for a crowdsourcing platform. Moreover, the crowdsourcing is both cheaper and faster than hireing experts.

1.4 Research Method

The research method employed in this thesis is the *Design Paradigm* specified in ‘Computing As a Discipline: Preliminary Report of the ACM Task Force on the Core of Computer Science’ [14]. This paradigm consists of four steps: state requirements, state specifications, design and implement the system and test the system. Stating requirements and specifications is done in Chapter 1 and Chapter 4, with design, implementation and tests presented and discussed in Chapters 5 through 8. The result is a real prototype with an evaluation consisting of a large set of experiments.

Two additional paradigms are presented in [14], namely *Theory Paradigm*, consisting of theoretical modeling and proofing, and *Abstraction (Modeling) Paradigm*, consisting of hypothesis construction and data analysis. Neither of these paradigms are suitable to our problem, because we seek to implement a functional prototype system, perform result validation and testing with Bagadus.

1.5 Outline

We introduce the Bagadus system with its existing capabilities in Chapter 2, including the camera array, ZXY tracker and the software useful for our video retrieval problem. Next, we explore the state of the art related to our problem in Chapter 3, where we discuss previous proposals to the action recognition and pose estimations problems. After introducing Bagadus and related work, we start by giving a deeper overview of the designed solution, presented as a series of programs that together forms the proposed workflow in Chapter 4. Then, in Chapter 5, the player tracking, including a frame normalization algorithm, is presented in detail. This is followed by the video annotation system that takes advantage of the crowdsourcing platform *Microworkers* in Chapter 6. In Chapter 7, the action retrieval algorithm is presented, which solves the problem of action recognition and pose estimation, and we present results to both classification and skeleton reprojection accuracy at the end of the chapter. Additional implementation details are then presented in Chapter 8, which contains implementation issues and optimizations found in our proposed solution, including the performance of the GPU port. We conclude the thesis with a conclusion and summary in Chapter 9, which also holds a discussion about future work and improvements that can be made to our solution.

Chapter 2

Bagadus

Bagadus [49] is a sports analytic system that combines camera images, sensor data and user input to record and log a soccer game. It aims at automating many of the tasks associated with sports analytic, including, but not limited to, player position, key events logging, video capturing and playback of all the obtained data synchronously. Other sport analytic systems exists, for example Interplay, ProZone and STATS SportVU Tracking Technology, but they usually require a lot of manual input.

Currently, Bagadus is installed at Ullevaal and Alfeim football stadiums, but it is adaptable to any situation. Professionals can use it to review games and aid in training, or consumers can use it for entertainment with additional statistical content. The primary features consists of panorama stiching, a virtual camera (a camera that can be moved withing a region of the panorama view), and search for video based on ZXY data and manual/expert's logging of key events. With its many capabilities and its potential for new innovation, it makes the perfect system for large scale testing for this thesis.

This chapter describes the part of the Bagadus sport analytic system necessary to solve our problem of content-based video retrieval, which includes the camera array, ZXY player tracking system and how these are used together.

2.1 Data Acquisition

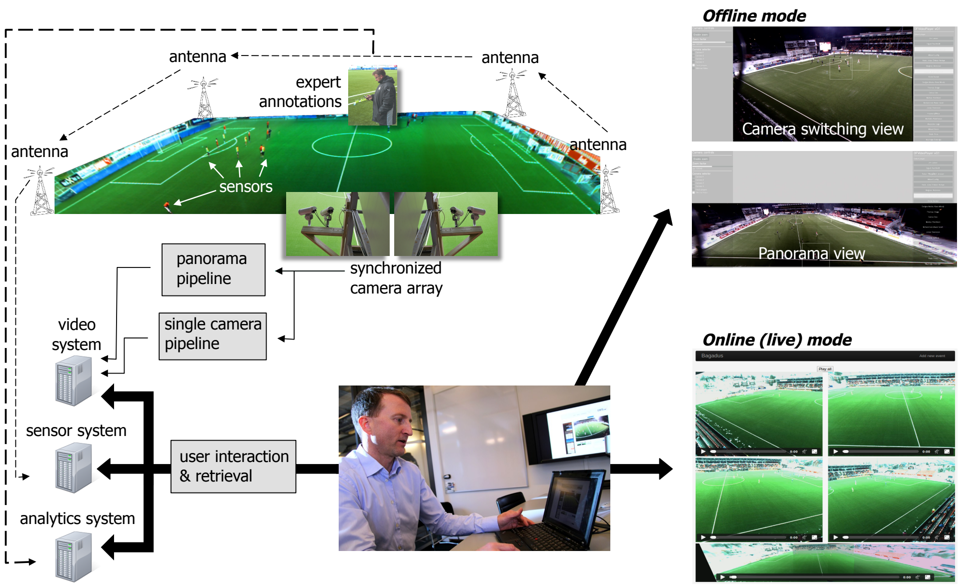


Figure 2.1: Overview of Bagadus and its systems. This includes the camera array and ZXY tracker system, the capture pipeline and the playback software. Playback supports both single-camera view and panorama (stitched) view.

Bagadus consists of three data recording modules: a video module, body sensors for all the players and user-entered event tags, as shown in Figure 2.1. The video module is a set of cameras that covers the entire field from a single array, with for every two cameras, there is a slight overlap to allow for stitching when generating a panorama view. The slight overlap is necessary in order to adjust the image from differences in viewing angles. Captured videos are encoded into 90 frames long clips, with timestamps of when the clip starts plus a clip number for that particular match.

Additionally, Bagadus has installed a player tracking module, which logs player positions (sampled several times per second) during a match into a database. Each sample has a timestamp that can be used together with the video clip time to pair up the two systems. More details about the cameras and positional system is described in Sections 2.1.1 and 2.1.2, respectively.

The third module is a manual logging tool. It allows coaches to take notes and enter key events during a match for later review. However, this logging system is of no direct use for our content-based video retrieval solution, but an important piece for any sport analytic system.

2.1.1 Camera Array

This section assumes knowledge about camera system and formats. It is recommend to read *Videologi* by Jacobsen [26] if more information on this topic is required.

The primary sensor module in the Bagadus system is an array of cameras, mounted centered above the tribunes on one of the long sides of the field. The camera array covers the entire field, with every camera placed in a semi-circle with a shared center point. The configuration of this array have changed over the years. We have used the setup consisting of three cameras with 1280×960 sensor resolution, but the setup is now updated to five cameras with $2.5K$ sensor resolution, with the same lens and camera make and model used within each array to maintain image appearance. The framerate differs between each setup as well, with the first system having 30 Frames Per Second (FPS) and the latest having 25FPS. Recording format is the same for all systems: planar YUV 4:2:0 or YV12 format compressed in a H.264 wrapper. The cameras delivers higher original image quality at 12-bit Bayer pattern, but this format is too data intensive and therefore requires compression. In this thesis, the older system with three cameras are used, as data is readily available and presents a higher demand for reliability and robustness to any algorithm to better test proposed solutions caused by lower resolution video.

When using multiple cameras it is important to have them all capture each frame at the same time. If not, then slight differences between camera pairs causes tearing between the overlapping image area in the panorama view and can potentially break the viewing experience. At the very least it causes cosmetic unpleasantness. Bagadus solves this problem by using a custom designed trigger-box that ensures synchronous capturing for every camera sensor [4].

2.1.2 ZXY Tracking Sensor

The radio tracking system is provided by *ZXY Sport Tracking* [61]. It uses antennas placed around the field to communicate with a radio unit placed on the belt of each player. The system captures each player at $20Hz$, recording positional information, as well as velocity and direction. Positional information is recorded with three axes - X, Y and Z -, but the Z axis is ignored since players do not move in this direction. The coordinate system follows the measurements of the field itself, where one unit of ZXY data represents one meter on the field. The exact field size and coordinates for some key elements are provided in Figure 2.2. Moreover, ZXY is accurate to within one meter of its actual system, making it extremely useful in many algorithms, including the ones developed in this thesis. The most recent installation from ZXY has improved the correctness of the tracking sensors down to 0.5 meters, but the older version is used in this thesis.

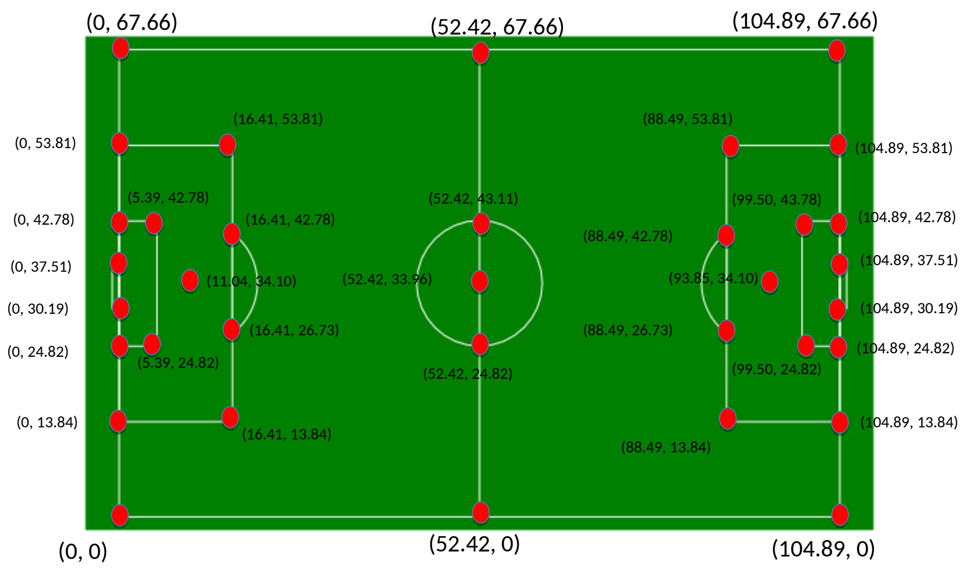


Figure 2.2: Key coordinates for the ZXY tracking system. The camera array is placed at the bottom center of the field, i.e., $(52.42, 0)$ [39].

2.2 System Synchronization

In order to use ZXY data with video captured from the cameras requires the two capturing systems to be synchronized. This synchronization must be performed on the timecode, i.e., a timestamp found on every ZXY measurement and video frame. However, these two systems do not operate at a frequency that can be matched perfectly against one another, that is, there is no one-to-one mapping between the ZXY timestamps and video frames. Instead, a $2:3$ ratio exists between the ZXY and video camera, meaning that interpolation would be required, if it where not for the two different recording types that allows for a non-interpolation method to be used.

Starting with ZXY, it takes, for every player, a new sample of its position, velocity and so forth. This information is then stored in a database with the exact timestamp when that sample was taken. The other system, video, is slightly more complex because of shutter speed. Shutter speed determines the exposure time (how long a frame is exposed) and is usually set to twice the rate of the framerate or higher, leading to exposure times less that the duration of a frame given a framerate. Even though this affects the amount of light captured for a frame can be less then a frame's duration, there is still a fixed number of frames per second. As of such, and for the sake simplifying the solution, a natural ordering of the two systems can be made: Every video frame is ordered into a sequence of time intervals, where each frame corresponds to $1/framerate$ of time, and the ZXY samples are plotted to the time-instance they are recorded. This yields to the natural ordering in Figure 2.3 between

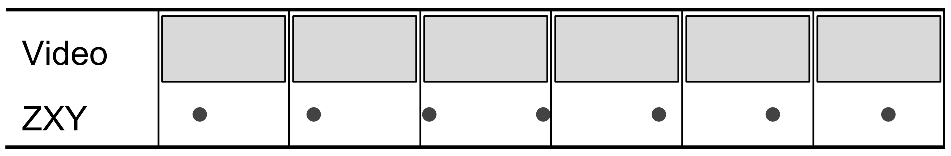


Figure 2.3: Synchronization between the video and ZXY inputs, with video showed as frames with duration and ZXY as sample points.

video frames and ZXY tracker data, without performing any interpolation or testing for when a frame needs to be used twice.

This method would work correctly, if it were not for a processing delay on the video, causing the video to arrive 3.4 seconds later that what the timecode specifies. To resolve this, a 3.4 seconds delay is added to the ZXY timecode to delay it (and thereby synchronizing the ZXY and video timecodes), because adding time to a stream is far easier than removing it. Another time difference could occur in earlier versions of Bagadus: It used separate computers for recording the two systems, which could cause drift in the timecodes caused by two different internal clocks. While this can be accounted for, as time drift is usually constant, it is much harder to determine and correct than a constant processing delay. Because of this, the current version of Bagadus performs both video and ZXY capturing on the same machine.

2.3 Field Coordinates to Camera Pixels

ZXY data is captured according to the flat playing field in 2D, but the camera observes the field from an angle. A 1-to-1 mapping from the ZXY data to camera pixels is not possible and a mapping process that performs the required position-to-pixel translation is required. One process to perform such an operation is homography mapping: It takes a set of pixel coordinates and the corresponding ZXY position, found by mapping the white lines on the field to pixel positions, as shown in Figure 2.4. Using OpenCV library's homography function to process the point-pairs, a matrix is obtained which is applied to the tracker coordinate tuple (consisting of x and y) to obtain final pixel positions. The homography mapping computes over eight possible axis's, which results in some errors inside the mapping and consequently causing points positioned outside the homography mapping to produce unreliable translations. However, once a homography matrix is obtained, it can be used without being recomputed unless the camera moves or another camera angle is used.

Despite the fact that the ZXY tracker is slightly inaccurate and that homography mapping can be a bit off target, performing only homography mapping works exceptionally well. Also, players have a height that could be a problem, because the ZXY position is on the ground, but with the camera

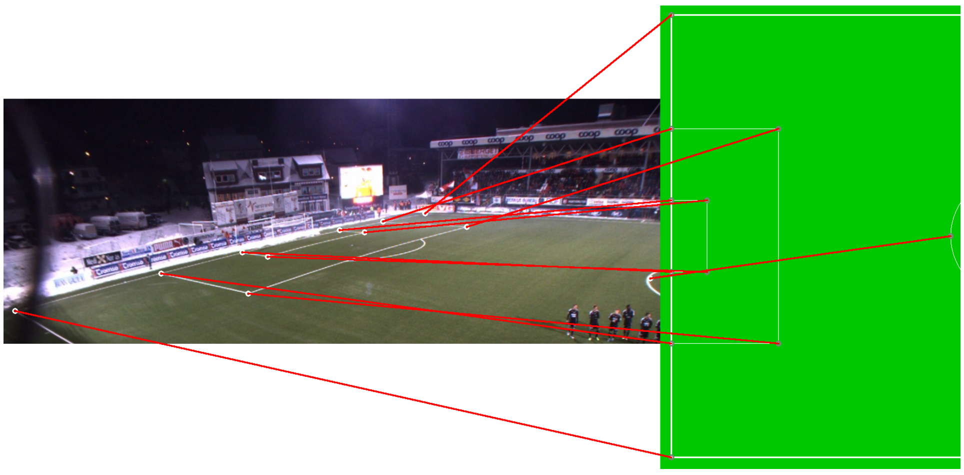


Figure 2.4: The mapping of ZXY coordinate system to the video image using homography mapping, using the white lines as rulers.

array placement looking mostly down on the field makes this issue immaterial. Bounding boxes can be drawn with the translated coordinate as the center, with 100 pixels in every direction. This bounding box will then encompass the player in most circumstances with the size taking into account small inaccuracies from both the ZXY data and homography mapping.

2.4 Summary

This chapter have described the Bagadus sport analytic system with its capabilities and capture devices: A camera array and a radio tracking system. The camera array consists of a set of cameras covering parts of the field, with slight overlap between neighboring pairs. A full panorama view can be stitched together or ZXY tracking position can be synchronized to overlay bounding boxes over players. Synchronization is done using the two systems' timecode without any interpolation, despite taking 20 samples per second for the ZXY tracking and having 30 frames-per-second in the video.

Bagadus is the system to which all captured player tracking data and video streams are obtained from, with the proposed solution designed to work with these systems. Consequently, the design of the player tracking algorithm must operate correctly within this enviroment, as all the video source material is obtained from Bagadus for use in both video annotation and classification. Moreover, it has an impact of the choice of methods that can be applied, as can be seen when looking into the state of the art in proposed solutions to action recognition and pose estimation.

Chapter 3

Related Work

The problem of identifying actions or poses using video images or other sensors have had an increase in interest in the past few years, mainly due to the large potential in data analysis, interactive applications (entertainment), surveillance, identification and control systems [1]. While most of the solutions to either of these problems are generally solved for a particular application or scenario, few solutions exist that attempt to solve the problem of action recognition and/or pose estimation for general usage scenario.

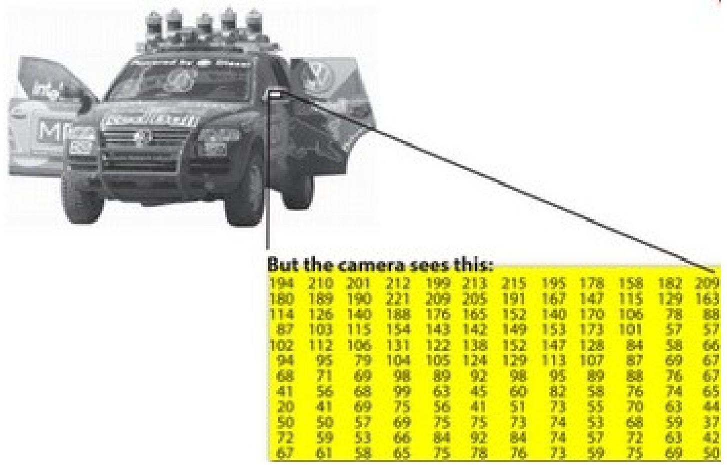
This chapter starts by briefly describing some of the concepts used throughout action recognition and pose estimation and continues with a discussion of proposed solutions for action recognition and pose estimation. At the end of this chapter, depth sensor based methods are discussed, and we argue for why this method cannot be used in Bagadus. Because there exists a vast set of proposed solutions, we mainly limit ourselves to describe the most recent works related to our problem domain.

3.1 Common Concepts

There are three concepts that are frequently used in action recognition and pose estimation methods, namely *Image Segmentation*, *Classification* and *Kinematic Constraints*. We therefore introduce these briefly here, before discussing the state of the art in action recognition and pose estimation.

3.1.1 Image Segmentation

Image Segmentation is the process of obtaining information about an image at a higher abstraction level than the one provided by the pixel values of the image. These abstractions can either be properties in the actual image or on a semantic level. For example, edge detection would define additional information found in the image from pixel intensities and would be a property of the image, whereas detecting cars would describe semantic information that is not available solely on the pixel values. In other words, image segmentation is a process where regions, e.g., a set of connected pixels, of an image are annotated with labels that describes *what* that region portraits. Depending on the image data and desired labeling, finding an object in a matrix of values is not necessary trivial, as Figure 3.1 illustrates.


But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	74	65
20	41	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Figure 3.1: Visual appearance and actual representation of a still image in a computer, illustrating the problem of grouping connected pixels to higher abstraction levels. Image taken from http://docs.opencv.org/_images/MatBasicImageForComputer.jpg.

3.1.2 Classification

Classification is a sub-field of supervised *Machine Learning* (ML) and is used to group input elements into different sets based on shared qualities or characteristics. Classification consists of three components: a classification algorithm, input data to classify and training data. The training data contains samples with ground truth, set by an expert in the field, to which the classification algorithm attempts to determine the mapping between samples and ground truths. After the classification algorithm has obtained a mapping from the trained set, it can provide a suggestion class for any new inputs. The solution is a *suggestion* rather than an absolute answer, since ML is a statistical framework generally based on hand-engineered data representations.

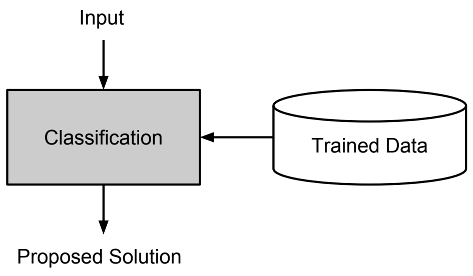


Figure 3.2: The design of classification algorithms.

A newer approach to classification, and other ML methods, are neural networks. It consists of a statistical learning model, inspired by biological neural networks and has proved to be both faster and more accurate on a variety of computer vision tasks. For instance, Tompson et al. [53] performed pose estimation using neural networks and estimated the pose faster and more accurately, compared to ML methods that do not use neural networks.

3.1.3 Kinematic Constraints

Kinematic constraints adds restrictions to a skeleton to prevent ill-formed poses from occurring in estimation results. Essentially, it applies rules to the joints' and limbs' degrees of freedom (DOF), i.e., the different ways these can be rotate and how long they can be. Limiting the DOF prevents skeletons from having a configuration that are physically not achievable by the human body. DOF is also used to describe the number of joints a model consists of, with each DOF modeling a single joint.

An example of a kinematic constraint and skeleton model that operates in 2D space, but which has the same movement limitations as if it were in 3D, is the Scaled Prismatic Model (SPM) presented by Morris and Rehg [35]. The SPM is built up as a series of chains with DOFs to both model and constrain individual joints and thereby the entire body. It is well suited to pose estimation algorithms that based on a single camera view that requires a model that complies with three dimensions.

Including kinematic constrains to an algorithm is not required, but it reduces the search space (possible poses) and generally improves the reliability and quality of the found poses, and is used extensively through proposed solutions in pose estimation.

3.2 Action Recognition

Action recognition, the problem of identifying what activity an actor performs, is considered as a classification problem, i.e., the one of obtaining a mapping between video sequences and action labels. From the taxonomy presented by Ronald Poppe [42], action recognition can be divided into two different ways of representation and a classification algorithm. The two representations are distinguished by the type of video *descriptor* they use (where a descriptor describes charateristics of a feature): *global representation* utilize the image data directly whereas *local representation* use interest points. Both global and local representations extract descriptors, i.e., feature vectors, that can be used by the classifier. The classification algorithm is independent of representation and is used for mapping video sequences to actions.

3.2.1 Global Representation

Global representations utilize captured video to extract the spatial and temporal information of an action. Consequently, a global representation is tightly connected to image segmentation, where background subtraction and optical flow vectors are good candidates for isolating and modeling the actor's movement.

Several proposed solutions for action recognition using global representations exists, such as the ones presented by Batra et al. [5] and Blank et al. [6]. These methods are almost identical and can be explained as follows: The input video sequence is first segmented into background and foreground using a background subtraction algorithm, keeping only the foreground elements. The foreground elements are then stacked on top of each other to form a three-dimensional volumetric space-time shape of the actor, with each layer representing a single frame and contains the spatial information and the layering technique encoding the temporal information (Figure 3.3). This shape can then be compared against a trained database to find a closely resembling space-time shape, and with it, obtain the query shape's action label.

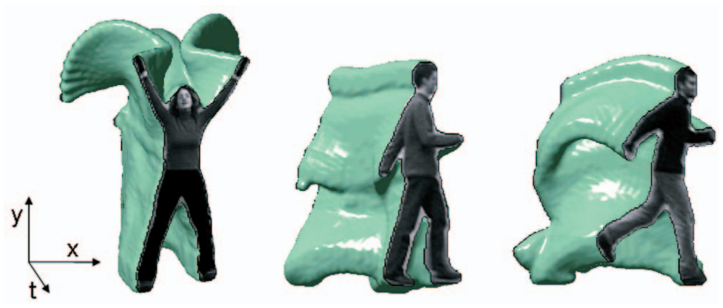


Figure 3.3: Space-Time volumetric figure of "jumping jack", "walk" and "run". Image taken from [6].

The difference between the solutions in [5] and [6] is that [6] places the foreground elements to form the 3D figure, whereas [5] uses shapelets, i.e., clusters of foreground elements grouped together by a defined threshold. In comparison, it yields to the same figure-like shape as the space-time volumes, but considerably bulkier. According to Batra et al., shapelets are more resistant to noise and can be represented as histograms, which again can speed up the classification process.

An alternative to volumetric shapes is optical flow. Efros et al. [15] rely on the optical flow calculated from the video frame pairs and uses the resulting vectors as the descriptors in the classification. With it, Efros et al. manage to obtain action labels with high accuracy, despite significant noise in the video.

3.2.2 Local Representation

Local representations are obtained by computing spatio-temporal interest points in a sequence of images. These points consists of spatial and temporal parameters, with the temporal one defining the relative movement between consecutive frames and spatial defining their locations within a single frame. Additionally, interest points are designed to be invariant to noise and viewing angles, which make them robust to challenging conditions. Moreover, being resistant to noise is particularly important for local representations, because the image is usually not segmented, e.g., foreground and background, that would otherwise have limited the points to only encompass the actor of interest.

One solution of computing the interest points is proposed by Laptev and Lindeberg [29]. The authors compute the interest points by detecting large variations in intensities of neighboring pixels in a video frame and storing the intensities' centroid as the spacial position in a feature vector (modeled as cuboids in Figure 3.4). The individual interest points' temporal parameter is obtained by finding the local maxima of points across frames, which also removes interest points that do not describe the motion. Furthermore, Laptev and Lindeberg locate the interest points by only using matrix operations and as a result never has to know the exact content of the image, which also reduces the required computational time and implementation complexity. Schuldt et al. [43] present an algorithm using spatio-temporal interest points for recognizing action, based on the interest points described in [29].

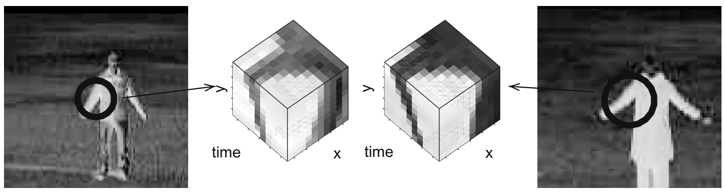


Figure 3.4: Representation of Space-time interest points as cubes, defining both position and frame changes . Image taken from [42].

One great advantage that interest points have over volumetric shapes is that they are invariant to camera movement and actor placement. Constructing volumetric shapes requires a consistent camera placement to properly capture the scene and maintain consistency, i.e., layering the frame correctly on top of one another requires the camera to be stationary. Non-centered placement can cause miss-classifications with volumetric figures, something that interest points does not suffer from. However, interest points are calculated from differences in video frames and could potentially find the wrong interest point in regards to the action performed, e.g., track a bird flying past in the sequence. As a side note, adding additional kinematic constrains could remove some of the incorrect interest points. Regardless of the representation method, the classification results deeply depend on the used classification algorithm.

3.2.3 Classification Algorithm

There are no restrictions to the classification algorithm to apply for a given descriptor, as long as it is able to perform mapping between the descriptors and the database entries. For example, in [5, 6, 15, 43], both Nearest Neighbor algorithm and Support Vector Machines are used.

There is one drawback of using a supervised classifier though, which is the database size limitation. The classifier will only work in the scenario it was trained for, e.g., it cannot label tennis actions if it was trained with only soccer players. However, the labels obtained for query sequences are usually quite accurate (they get correctly labeled) in the above-mentioned papers.

Action recognition is a classification problem, where the quality of classification results mainly depends on segmentation and the classification algorithm, whereas pose estimation requires more work in order to label every single joint and limb for every single frame.

3.3 Pose Estimation

The already existing pose estimation solutions can be divided into two different classes as presented in the taxonomy by Poppe [40]. The two classes are *Model-Based* and *Model-Free* which are differentiated based on whether a human model is used in the estimation or not. Specifically, model-based heavily relies on having a human model before any estimation can be done, whereas model-free builds a pose from the images without a human model.

3.3.1 Model-Based

Model-based pose estimation requires a model of a human which is then matched to an image. The human model can be defined with any number of DOF and supports either 2D or 3D environments with optional kinematic constraints. A frequently used model is the Scaled Prismatic Model (see Section 3.1.3), that operates in 2D with 3D constraints and is a good solution when using a single camera.

Another single-camera 2D pose estimation algorithm using rectangular or trapezoid-shaped patches for modeling humans is proposed by Ramanan and Forsyth [41]. Here, the authors segments the image into texture patches that is kept in a list over possible limbs. With these patches, along with a human model with kinematic constraints, a pose is constructed by connecting the patches into human-like figures that defines the pose. One drawback with patches is that they have a relatively bulky form, which at times might not be detailed enough for certain applications. Furthermore, having the pose represented in 3D is usually more desirable.

Performing pose estimation in 3D generally requires a set of cameras recording the same scene from different angles and then having the body-model placed in 3D space in a way that ensures it matches up with every single camera view. Usually, every video stream is run through a background subtraction algorithm to obtain a silhouette, which makes it easier to ensure the exact placement of the actor in 2D/3D space [16, 19, 59]. The human model can then be matched into the silhouette and adjusted accordingly for each frame. Alternatively, the shape of the actor can be triangulated from the silhouette to obtain a full outer hull, a method referred to as visual hull reconstruction [12], that builds up the shape one voxel at a time. A voxel is a small cube that covers a tiny amount of space with a color, not that dissimilar to a 3D pixel.

The proposed solution by Zhang et al. [59] uses several cameras with visual hull reconstruction from silhouette images to obtain a pose. They do this by placing and adjusting a human model iteratively until it aligns perfectly with the voxels and confines to the kinematic constraints. Gall et al. [19] take this one step further and extend the model with a mesh that defines the surface of the actor, as well as providing a 3D skeleton. In both [59] and [19], the algorithm runs in two steps: First, they try to find all possible model configurations and secondly they try to find the best model and adjust any small misalignments that might be present. The result is a high-quality 3D skeleton and (optionally) an outer mesh (Figure 3.5), but at a considerably high computational time as the voxel shapes must be reconstructed for every single frame.

Having a model when performing pose estimation is beneficial because it can be used for asserting correctness of a pose (e.g., the leg is a leg and not an arm), discarding impossible poses, reducing the number of iterations needed to match poses and images and prevent random objects in the video from becoming limbs. Unfortunately, the model describing the human pose might not be fitting for all actors, e.g., [59] must adjusts the model for different people. Using an approach without relying on a model can then be more desirable.

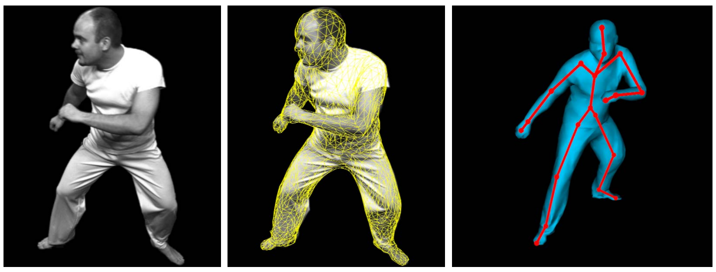


Figure 3.5: Original image (left) with the mesh (center) and skeleton (right) next to it. Image taken from [19].

3.3.2 Model-Free

Model-Free solutions do not use a human body-model to obtain the pose, but instead relies on a direct image-to-pose relation. Elgammal and Lee [16] propose a model-free pose estimation algorithm that models the actor as a silhouette, with every silhouette pixel having an additional value defining the distance from the nearest silhouette edge. These distance values are then modeled as a matrix that is processed using manifold learning. Manifold learning locates the key interest points among the silhouette distance values and encodes them into a lower dimensionality descriptor. This descriptor is then used in a classification algorithm to obtain skeletons from a trained database. Elgammal and Lee also use manifold learning to construct the trained database skeletons, using the interest points as skeleton joints. An alternative to manifold learning is use of Principle Component Analysis (PCA) [21], as it also locates the key interest points and encodes them into a lower dimensionality for classification. Figure 3.6 shows an example action from different views, where the interest points are represented.

A simpler method is proposed by Efros et al. [15] where the authors use a trained database to reproject skeletons with a similarity measure based on optical flow to match trained and query sequences to reproject skeletons into the query sequence.

As with the action recognition problem, and unlike model-based approaches, model-free approaches are limited to the scenarios where a same-class entry can be found in the database. The database can be extended, but it will still be limited to the existing and extended scenarios.

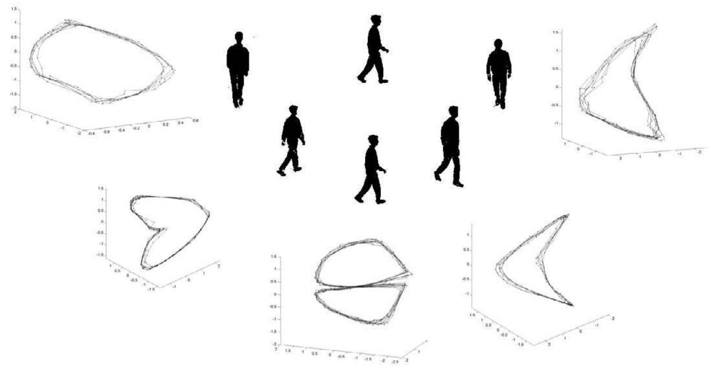


Figure 3.6: Manifold learning results showing the walk pattern from different angles. Image taken from [16].

3.4 Pose Estimation with Depth Sensors

An alternative to using video cameras to capture an actor's performance is using depth sensors: non-intrusive devices that records depth information rather than color intensities for every pixel. A Time-of-Flight (TOF) camera is a depth camera that measures the distance between itself and the subject (e.g., actor) by measuring the time light-waves spend on traveling between the camera and the actor. The distance can then be found by converting the time info based on the lightspeed.

Several solutions to pose estimation using TOF cameras have been proposed and some of them are presented in this section. We will also mention the Microsoft's Kinect, a device for registering motion (i.e., pose estimation). However, the solutions based on TOF cameras cannot be used with Bagadus, because Bagadus does not have TOF cameras installed. Installation of TOF cameras is not an alternative either, as cameras usually has a severely limited range of four meters or less and resolution of QVGA or lower [25]. More expensive systems exists with longer ranges and are used in avionics and space applications [11]. Either way, TOF cameras are unsuitable for the Bagadus system due to cost or hardware limitations, but when the technology eventually improves it might become an alternative for the Bagadus system and is therefore included here for comparison purposes.

3.4.1 Time-of-Flight Algorithms

TOF cameras' aquired data can be modeled as a 3D cloud, with each distance measurement defining a point in this cloud that can be restructured into a useful shape for pose estimation (see Figure 3.7). Ganapathi et al. [20] propose a statistical approach that models each depth measurement in the cloud as a position and velocity pair into a graph. This graph is then used to compute the most likely interest points using Dynamic Bayesian Network, with the highest probability points sent to a classifier. The classifier takes these interest points and attempts to map them to limbs, and if a relatively certain mapping is detected, it is marked as a found limb. This is then done until all limbs are found or the data-points are exhausted, in which case remainder limbs are considered obscured.

Another approach by Baak et al. [3] describes an algorithm that starts by creating a path through all the sensor's data points. Using weighted Dijkstra algorithm to compute the outermost hull of the cloud, it constructs the first pose hypothesis out of the most likely interest points found on the path. Then, it queries a trained database with the most outlying points of the hull to obtain the second pose hypothesis. The construction of the database is done with a marker-based motion capture system, which is then normalized to make the database match up to as many query sequences as possible. The two hypothesis' are then fused together to provide the final pose, with the different hypothesizes used to ensure quality and provide robustness to the random noise found in the recorded TOF cloud.

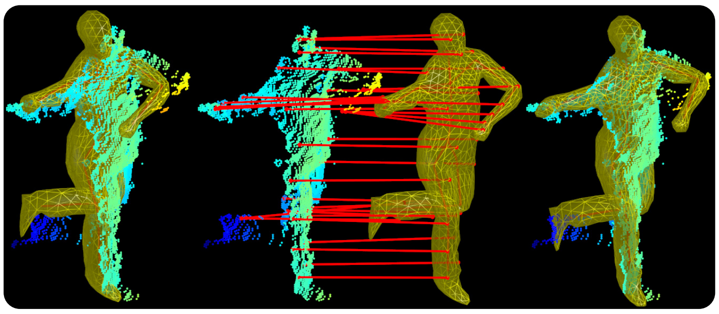


Figure 3.7: Pose estimation using TOF clouds, with leftmost an initial match, the center showing the mapping between TOF cloud and pose and rightmost the final adjusted pose. Image taken from [3].

3.4.2 Microsoft Kinect

Microsoft Kinect is designed as a home entertainment system to be paired with an Xbox console. It obtains poses that can be used with games and other interfaces and can be extended to other applications using the Kinect SDK. Because the Kinect is a proprietary device, only the algorithm provided by Microsoft is the topic of this section. The algorithm implemented by Microsoft [45] is able to estimate poses in real-time using a combination of TOF sensors and classification.

More specifically, the Kinect consists of a set of sensors, including a video camera and a IR sensor and emitter that operate like a TOF camera [27]. Using the IR emitter and video camera, a fast and robust pose estimation algorithm based on classification has been developed. Because Kinect is assumed to operate as an entertainment device in a living room, Microsoft were able to construct a complete classifier for any type of scenario and represent it internally as a tree. A tree-structure for poses allows for both fast estimation and quick validation of the pose. Without a doubt, the Kinect is probably one of the more accurate and reliable pose estimation devices for commercial use, but is unfortunately very limited in its use to living room sized scenes and only a handful of people.

3.5 Relationship between Action Recognition and Pose Estimation

The two problems of action recognition and pose estimation are quite similar, with some approaches utilizing the same technique, e.g., background subtraction combined with classification. Moreover, they both seek to obtain an actor's performance, but with slightly different results, i.e., action label contra skeleton. It is possible to solve one of the problems and thereby the other, in other words, obtaining the skeleton and using it as input for action classification, and vice versa. For example, Stone and Skubic [50] perform visual hull reconstruction to detect when an elderly person is lying on the floor, which can be considered as the two classes "on floor" and "not on floor". Also, Efros et al. [15] use action classification as a first step towards obtaining the skeletons. In [15], query sequences are classified based on motion and when the correct query action is found in a trained database, it is used to reproject the skeleton onto the query sequence. Inversely, in [54], the pose is first recovered which is then classified using a SVM to obtain actions. The proposed methods by [15] and [54] are both robust and accurate, but also shows how similar the two problems of action recognition and pose estimation are.

3.6 Summary

This chapter started by introducing different concepts commonly used when discussing related works om action recognition and pose estimation. A variety of proposed solutions exists for both problems, but without any definitive solution that solves either of the problems.

Proposed solutions for Action recognition can be divided into two forms of representations and also based on the classification algorithm used. The two representations are either global representation or local representation, depending on whether the video frames are used directly or not. An additional classification algorithm is used to map representation to actions, for example with KNN or SVM.

Likewise, the pose estimation problem can be divided into two classes: model-based and model-free. In short, the model-based is heavily reliant on a human body-model (e.g., SPM) to perform model-matching into an image. The results are robust and accurate, but suffers from excessive computational time caused by having to reconstruct the 3D voxel shapes from camera angles for every single frame. On the other hand, model-free does not rely on a body-model at all, but instead utilizes statistical methods or direct frame-to-frame similarity matching. However, the pose model-free solutions are usually limited by the database they query for skeletons.

One do not need to use video cameras only, but motion capture devices and markers are too intrusive, and therefore disregarded. Another type of sensor exist for capturing depth, called Time-of-Flight cameras, but they are far too limited in range, resolution and are too expensive to be usable in the Bagadus use-case of a large soccer stadium.

There are many different solutions for both action recognition and pose estimation, with some of them being quite similar to one another. Moreover, some of the solutions attempts to solve both the action recognition and pose estimation problems at the same time. The most notable of these are the one by Efros et al. [15], which uses optical flow and a matching function for both classification and skeleton reprojection.

Which one of these methods that are most suitable to our problem domain, i.e., most suitable with the Bagadus system, depends on the limitations and assumptions found in Bagadus (see Section 1.2.3). In short, the imporant limitations are low resolution video, high levels of noise and only a single camera array. The biggest consequence of having only a single camera array makes visual hull reconstruction or any other model-based solution infeasable, as they require more than one viewing angle. Using a model-free approach is a better choice to obtain poses, as it does not require multiple camera arrays. Moreover, this approach is quite similar to local representations for action recognition, as they both rely on interest points, but obtaining good interest points in the low resolution and noisy video for both problems might be unsound. Inversely, by using a global representation and a classifier to first obtain actions and then use the obtained action sequence to estimate poses using a model-free estimation approach is much more viable, as this requires only a single camera and can work with low resolution video sequences.

A solution based on a model-free and global representation is the solution proposed by Efros et al. [15], which solves both the action recognition and pose estimations problems, as discussed in Chapter 1, and briefly revised in this chapter. Furthermore, the proposed method by Efros et al. addresses a very similar problem compared to ours and we therefore chose this as the basis for our proposed solution, presented in the next chapter.

Chapter 4

System Overview

Before getting into the details of content-based video retrieval, player tracking and the final pose estimation process it is important to have a an overview of the system which has been designed in this thesis. This design consists of a workflow of programs, with each program designed to solve one of the problems described in Section 1.2.4, as well as some programs to either supplement or aid in data management (particularly for crowdsourcing data). This chapter covers the overview of the proposed workflow, programs and its integration in Bagadus.

4.1 System Layout

The designed layout is meant to work well together with the existing Bagadus systems, as well as being reconfigurable and adaptable to any new changes in algorithms or research results. The remainder of this chapter includes a description of our proposed prototype solution and how it relates into solving the three procedures defined in Chapter 1.2.4. For more implementation details see Chapter 8 and for the source code check out [24].

4.1.1 Data Files

The data format used in this thesis consists of `png` images, `csv` files (Comma-Separated Values), `SQL` files containing SQL queries and regular `txt` files for additional program control. This is done in order to have consistent inputs and outputs across all the different programs, but also because the original data used in this thesis comes from CSV files [39]. The only file not being either an image or a CSV file is the serialized file created by the `Data Generator`, which is a binary packed vector field of optical flow vectors.

4.1.2 Programs

There are a few implementations required in order to realize a content-based video retrieval prototype. We have decided to implement different programs, with each program assigned a particular functionality. All of the programs listed here and the *Skeleton Estimation workflow* (Section 4.1.3) are used to solve the problems presented in Section 1.2.4, namely video extraction, video annotation and action retrieval, and can be directly related with the following programs:

1. **Video extraction** is solved primarely by the `Data Generator` program, as it is reused as a component in the `Motion Classifier` program when it reads query sequences, and is described in detail in Chapter 5.

2. **Video annotation** is solved by the `Online Training Tool`, and is described in detail in Chapter 6.

3. **Action retrieval** is solved by the `Motion Classifier` program, and is described in detail in Chapter 7.

Data Generator

Extracting video sequences to be annotated or used in classification and skeleton reprojection is done using the `Data Generator` program. The `Data Generator` program takes as input a set of video clips (captured by the Bagadus system) and a CSV file containing the exported ZXY database (ZXY file). It then produces a set of images and an SQL file, specified by the ZXY file, for use with the `Online Training Tool`. In addition, it computes the optical flow for the exported images into a loss-less compressed file, which can be used by the `Motion Classifier` program to obtain classes.

Online Training Tool

Unlike Efros et al. [15], we do not have access to skeleton data (joint positions) for captured video sequences and must therefore annotate our own dataset. This is done in an online tool, a web interface that is used for annotating the images generated by the `Data Generator` program. The annotation task in the online tool includes placement of joint positions and action labeling, as well as selecting a bounding box around the player. All the data are stored in a database, which can be exported to CSV files and feed the `Result Checker` program.

Result Checker

The `Result Checker` program is an utility program that have several functions for validating and visualizing data. It primary function is to assert and filter data annotated with the `Online Training Tool` and merging them into a single CSV file (as each image will have several annotations that must be fused to a single point). This can be done by using either Gold Standard Tests or Majority Voting, described in more detail in Chapter 6. Additionally, it can be used to examine the intermediate CSV skeleton files or individual worker's click-point performance by rendering skeletons into images, which is used to obtain the figures in Chapter 7.

Motion Classifier

The `Motion Classifier` program takes the output generated from the `Data Generator` program plus the merged training data from the `Result Checker` program and uses these to classify a query sequence to obtain a class label and perform skeleton reprojection based on trained data. In other words, this program contains our proposed solution for the content-based video retrieval problem. The produced output is a video sequence with the skeleton projected onto every frame and a CSV file with the exact joint positions.

Coordinate Translator

The `Coordinate Translator` can be used on the exported annotated data to obtain absolute pixel positions for use with external programs (or with any of the intermediate results produced by any of the other programs). It is required to translate the trained joint locations (or any of the positions between programs) to absolute pixel coordinates as they are given as floating point numbers relative to the tracker region of interest (ROI) and worker-selected bounding box. The only case where translation is not needed is with the final skeleton output, which already comes as absolute values. The exact details for this translation can be found in Section 8.3.1.

4.1.3 Program Workflows

In this section, we present three main workflows combined using the different programs listed in the previous section, with the individual pipelines for every program described in Chapter 5 through Chapter 8. We differentiate between workflow and pipeline based on the definition from [56] with slight adaptation to the problem addressed in this thesis:

- **Workflow**: A set of programs processing a video sequence in order to obtain a pose or an action label. Each program can be used zero or more times in a workflow.
- **Pipeline**: A linear sequence of specialized stages that is used to build up a program in order to solve a particular problem. Each stage must only be used once in a pipeline.

The three primary workflows are named *Skeleton Estimation*, *Crowdworker Quality Control* and *Skeleton Validation*. These workflows contain everything from extracting bounding boxes to exporting skeletons, validating crowdworkers performance and verifying skeleton outputs from different programs. Additional programs can be inserted into the existing workflows, as long as they comply with the input and output formats using CSV, or new workflows can be created if required. However, these named workflows are the ones used to obtain the results presented in this thesis and is presented here to give an overview of how the different programs interact.

Skeleton Estimation

Skeleton estimation is designed to solve the problem of content-based video retrieval of soccer players from video captured by the Bagadus system and annotate them with an action label (the workflow is illustrated in Figure 4.1). It utilizes almost every single program, starting with `Data Generator` to extract images from a set of video sequences, recorded by Bagadus cameras and ZXY tracking sensors. The exported images are then used in the `Online Training Tool` to be annotated with skeletons and inserted into a database. This database is then exported as CSV files, containing the annotations and frame identification numbers. Using the `Result Checker` program, the crowdsourced annotations get merged to produce a single skeleton per frame. By putting this merged file together with the serialized file generated from `Data Generator`, the `Motion Classifier` estimates the most similar annotated sequence for a query sequence and exports the resulting skeleton to a CSV file. Optionally, a rendering of the skeleton can be provided as a series of images. The annotated CSV skeleton for the query sequence could then be fed to other computer algorithms to, e.g., perform scene reconstruction or to perform additional semantic analysis and statistics.

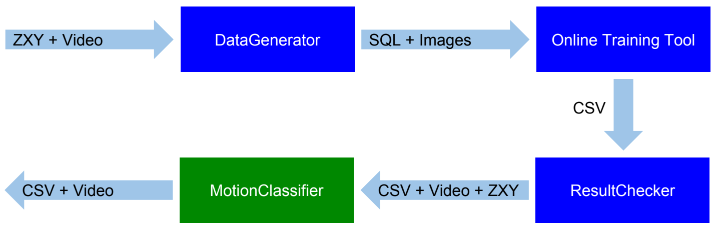


Figure 4.1: Entire workflow designed to obtain skeletons via content-based video retrieval and crowdsourcing.

Crowdworker Quality Control

Checking the quality, or accuracy, of the workers from the crowdsourcing campaign is done in the `Result Checker` program. At the same time, the merging of the training is calculated, and exported as an SQL file containing the quality measure (dependent on merge type). This file can be used to accept or reject the annotators' effort, if the work is a paid contribution. The workflow in Figure 4.2 displays both of the outputs generated from the `Result Checker` program, as well as the workflow to obtain the skeletons in the first place. The merged CSV file can be used to examine the combined effort of the annotators.

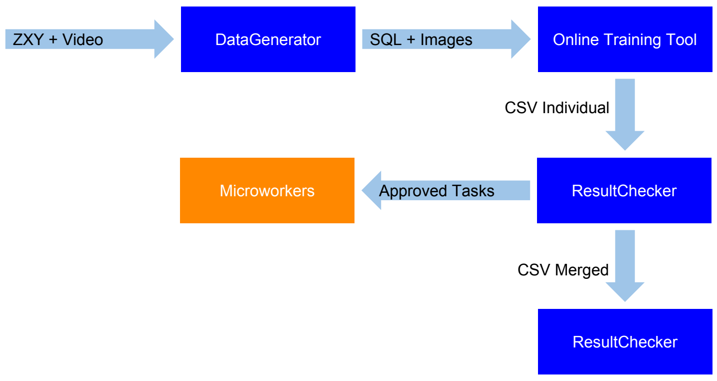


Figure 4.2: Crowdworker Quality Control Workflow. CSV Merged consists of individual annotations, whereas CSV Merged consists of the merged results from all the annotations.

Skeleton Validation

Many of the programs listed above outputs skeleton data as the only result or as a result of some computation. These skeletons can be viewed using the `Result Checker` program to perform manual inspection and validation of the joint positions. The program can be used as illustrated with red arrows in Figure 4.3.

It is particularly useful in viewing the skeleton annotated by crowdworkers, as it will help in identifying problems, but also for analyzing reprojected skeletons from `Data Generator`.

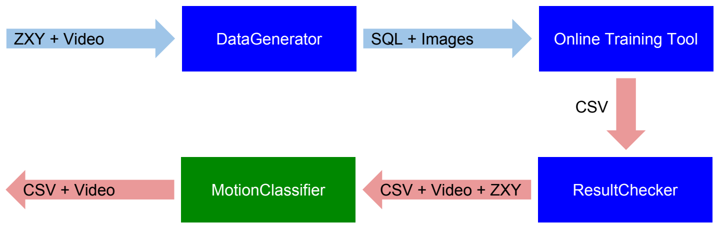


Figure 4.3: Workflow for obtaining skeletons, where red arrows indicate intersection points where the skeleton can be verified.

4.2 Summary

In this chapter, we have presented the programs and workflows designed to solve the problem of content-based video retrieval. The set of programs are made to solve each of the three procedures introduced in Chapter 1.2.4. With these, we create a modular workflow that can easily be incorporated into the existing Bagadus system, by using a common input/output data structure with CSV files. An additional set of utility programs and workflows aid in managing and verifying results produced during the different stages, as well as performing quality control on annotated training data.

In the following chapters we present the three procedures/programs for Player Tracking/`Data Generator` in Chapter 5, Video Annotation/`Online Training Tool` in Chapter 6 and Action Retrieval/`Motion Classifier` in Chapter 7 with additional implementation details in Chapter 8.

Chapter 5

Player Tracking

Encoding video sequences of soccer players as feature vectors for classification requires a tracking algorithm that isolates and size-normalizes the individual players. Isolating individual players are required in order to ensure a successfull comparisons of actions independent of where the player is located on the field.

This chapter presents the pipeline for tracking soccer players and extracting small player-centered images based on background subtraction and data from the ZXY module. Additionally, the pipeline constructs precise bounding boxes around the player that far exceeds the current system implementations. The pipeline for tracking players solves the video extraction problem, but not for any given situation. Therefore, we also discuss further and future improvements and open issues at the end of this chapter.

5.1 Player Tracking Pipeline

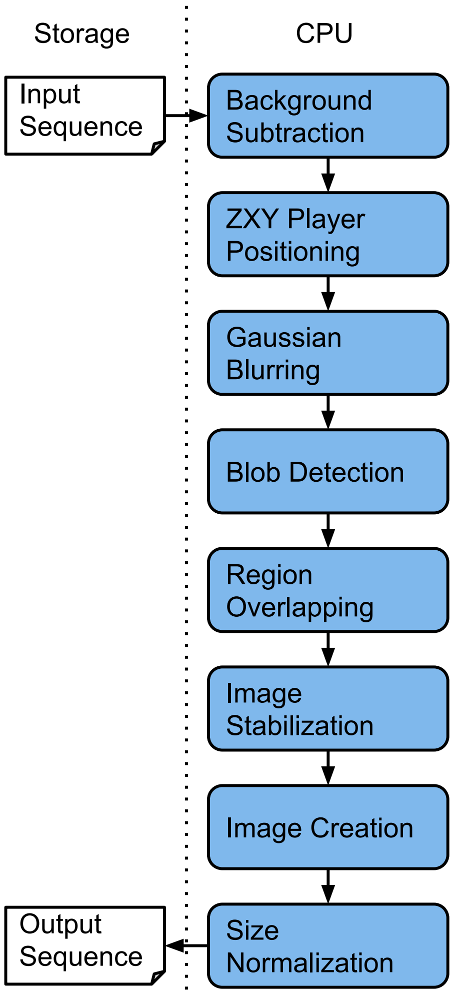


Figure 5.1: The player tracking pipeline, with data in white and CPU stages in blue.

There exists several other tracking algorithms for identifying objects and follow them during a game [57], but because we have access to player position data (from the ZXY sensors, see Section 2.1.2) we can use this data instead of having to find the players on the field first. Subsequently, by using the ZXY data, tracking players and exporting normalized images is then equivalent to finding the region on the field that a player occupies, refered to as the *Region of Interest* (ROI), and center the player within this ROI. ROIs can be viewed as a bounding box (rectangle) surrounding soccer players on the field and are used interchangeably in this chapter. With an obtained ROI, the player within this ROI is size-normalized to better support comparisons. The final ROI is exported as a grayscale image with background removed in order to support OpenCV's optical flow algorithm and to remove uninportant data (e.g., grass). All of these steps (i.e., tracking, centering, background removal etc.) are implemented as stages in a pipeline, each of them being presented in two parts describing how it is used and how it is done. A complete overview of the pipeline is shown in Figure 5.1. Each of the steps are explained in the subsequent sections.

5.1.1 Background Subtraction

The first stage in the pipeline performs background subtraction on the entire video frame to define which pixels is located on an imaged soccer player. Background subtraction is performed on the full frame, rather than on the region on interest, to prevent the white lines on the field from being marked as foreground. The white lines would be marked as foreground if the ROI is used, as the lines will appear to move across the frame and thereby be considered foreground.

How it is Used

The goal of background subtraction is to separate an image into two planes: foreground and background, where the foreground encompasses all the pixels that are of interest (as in Figure 5.2). The foreground plane is also referred to as a mask, since it marks out the pixels of interest. In our case, the interesting pixels consists of the soccer players. Visually, background subtraction gives a filled outline of every player, as shown in Figure 5.3 on page 38, and is particularly useful when there is a need to identify objects in a scene where the background is irrelevant. Consequently, the foreground mask is used several times through the player tracking algorithm to obtain the region of interest. Other ways of finding the players in a video sequence exists, e.g., edge detection or Hough Transform, but a background subtraction algorithm is the most suitable in this use-case: It provides a per-pixel object separation and high quality masks because we have a static camera and it is used with great success with other solutions in the Bagadus system [2].

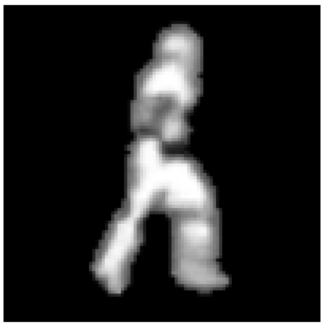


Figure 5.2: A grayscale image of a player with the background removed, used for computing optical flow.

How it is Done

Every background subtraction algorithm has the same basic idea: An initial picture is set to define the static background and every succeeding image is then compared to this one. If a pixel differentiates in intensity or color compared to the static background, it will be considered as foreground. More advanced algorithms extends the basic concept to be more robust to changes and noise, such as adaptive algorithms than learns the difference in foreground and background as time progresses. For instance, a more robust approach to background subtraction is proposed by Cheung and Kamath [13].

The applied method to subtract the background in this thesis is the MOG2 algorithm, provided by the OpenCV library. It uses an implementation of Gaussian mixture-based background/foreground segmentation, a configurable and adaptive algorithm. It must be tweaked to every lightning condition, but doing so makes it easier to obtain the correct mask. MOG2 has been used since first installment of the Bagadus system and have proved both accurate and adjustable to the environmental challenges provided by an outdoor stadium [2].

What is particularly interesting with MOG2 is its ability to learn changes in the background. Every consecutive frame processed for background subtraction is also added at the front of the learned background history. In short: the newer the image is, the more it affects the decision on which pixels are foreground and background. This history replaces the single static initialization image and offers an advantage over other algorithms when the scene slowly changes. MOG2 also applies a Gaussian Filtering of pixels for a given foreground/background computation, making it more tolerant to noise. This is extremely useful for Bagadus, with outdoor matches having clouds changing the ambient light slowly but regularly. The conjuction of learned history and Gaussian filtering makes MOG2 also able to distinguish out shadows to a certain degree. While potentially useful, shadow detection is not used in this thesis.

5.1.2 ZXY Player Positioning

Background subtraction produces a set of foreground *shapes* (a set of connected foreground pixels that marks out a soccer player) that marks the location of all the soccer players. However, only a single soccer player is to be extracted at a time and a selection of one of the foreground shapes must be made.

How it is Used

The ZXY sport tracking module found in the Bagadus system contains a radio tracking device for every soccer player and can be used to identify the foreground mask objects. Over the course of a soccer game, a single player can be pointed out in the video based on ZXY data, which is used to obtain an initial ROI. Other algorithms for following a player's silhouette can be used, but the ZXY data provides a consistent and accurate tracking.

How it is Done

By performing homography mapping (Section 2.3) between the camera view and ZXY coordinate system, a position within the video frame is obtained for a player's position that can be used to create a rough bounding box. This is the existing solution for identifying players, but it has a much a larger headroom in the bounding box compared to an optimal bounding box. Although, it is a good starting point for narrowing the box based on the foreground mask provided by the MOG2 algorithm.

5.1.3 Gaussian Blurring

The MOG2 background subtraction is not perfect and will produce foreground masks for background pixels (also called false positive, see Section 7.3.1). Such errors can be divided into two different types that we refer to as *single pixel errors* and *blob errors*. Single-pixel consists in a single foreground pixels not connected to any other foreground pixel and usually occurs in large changes in the grass. Blob errors is a group of foreground pixels connected together to form a larger silhouette. In this pipeline, single-pixel errors are removed by applying Gaussian Blur (i.e., Gaussian Filter).

How it is Used

A single pixel error is a single pixel marked as foreground, while it is actually a background object. It is usually a result from slight pixel variance on the playing field caused by moving grass or similar. This error is removed by applying a standard Gaussian Filter to the obtained foreground mask. It even comes with a desirable side-effect: the silhouette's edge of a player object gets a margin of error. Because a single pixel can contain both a player and background elements, it becomes uncertain whether the pixel is foreground or background. Having the margin ensures all of these border-type pixels are included as foreground, which ensures as much as possible of the small players are included in the image creation later on in the pipeline.

How it is Done

Gaussian blur is one of the simplest operations in image processing: Its function results in blurring an image by combining the values of neighboring pixels via convolution with a Gaussian Kernel (a matrix defining the blurring size and weights). Despite its simplicity, it is extremely useful in combination with other algorithms. Its uses extending from noise removal to softening high contrast values (edges) and more. Noise removal is particularly important, both in terms of poor quality images and output values. Even the most sensitive cameras are subject to noisy measures and Bagadus' camera array is no exception. By applying Gaussian filter, it is possible to reduce the noise at the cost of slightly less sharp image and information loss caused by merged values. Secondly, image processing algorithms can introduce new or additional noise in the output, to which a Gaussian Filter can be applied to reduce the error factor. Unfortunately, it cannot blur away erroneous foreground blobs.

5.1.4 Blob Detection

Using the foreground mask and initial ROI provided by the ZXY trackers, the ROI can be compressed tightly around a player. Given that no blob errors are present, it would be a decent bounding box. When that is not the case, a selection must be made to cut out the stray blobs.

How it is Used

Blobs consists of larger collections of foreground connected pixels, often coming from the shadow of players. The MOG2 algorithm attempts to some degree to deal with shadows, but is not always correct. Filtering out blobs is more difficult compared to single-pixel errors, as it could potentially be a player, but can be done using a blob detection algorithm. Observed instances of video sequences shows that the erroneous blobs are smaller than the ones for the players and the smaller blobs are weakly connected, if connected at all, to other blobs (see Figure 5.3). This can be exploited by running the frame through a blob detection algorithm and calculate the blobs' area. Whichever blob that is largest is assumed to be the player (which is almost always the case) and the remainder small blobs can be erased. Using this approach have the advantage of only having to count pixels rather than analyzing them, which both speeds up and reduces complexity of the tracking algorithm.

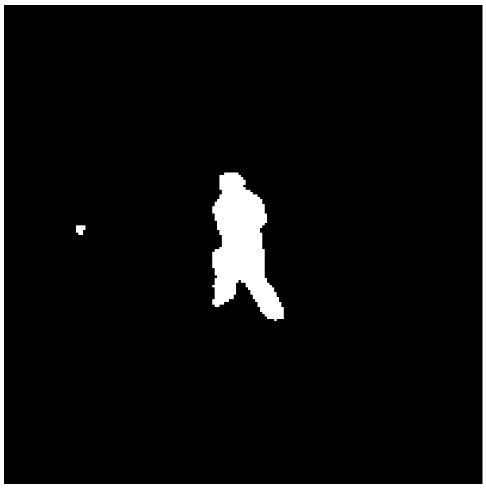


Figure 5.3: Foreground mask containing two blobs, with the leftmost (smallest) being an error and the rightmost (largest) the player.

How it is Done

Blob detection consists in segmenting the frame into different regions of connected foreground pixels and group them together, or simplified as the problem of finding outlines of shapes. OpenCV provides a function to detect contours (i.e., outlines), which is equivalent to finding the edges around each blob. It does this by finding a foreground mask pixel placed at the border of a blob and then follows the border pixels around the blob until it arrives back at where it started. This process is repeated for every single blob in the image, resulting in a set of blob contours. The blob with the largest areal, based on the contour, is selected and a bounding box is placed around it, unless several players (large blobs) are in the image.

5.1.5 Region Overlapping

If two or more players are present, then there is a problem to decide which one of them to track. Although sequences can be selected from a game to only have one player in the ROI, it is not always feasible, because the ROI provided by the ZXY homography mapping is required to be large (i.e., 200×200 pixels) making it nearly impossible to avoid the presence, if only parts of, other players.

How it is Used

Desciding between, e.g., two players, inside a single region of interest could be done by querying the ZXY tracker data and figure out which of these players to track, e.g., by knowing which is to the left and which is to the right. In this way, a selection can be made based on the placement of the players within the ROI, but this solution is not always feasable. For instance, the player on the opposite team do not, in our dataset, have ZXY motion sensors on them. Furthermore, an experiment with making the ROI closer to the ZXY tracker position gave the bounding box a tendency to jiggle at random times, contradicting the desire for a stable image. Instead, under the assumption that a player cannot move faster than the limits of the previous frame's bounding box, we check that the next selection we are attempting to find does overlap as much as possible with the previous ROI. With enough overlap, one can be certain (under our assumption of movement speed never exceeds the ROI) that the two ROIs encompasses the same player. In short, a player is tracked by having *sticky* bounding boxes, i.e., bounding boxes that prefers to move as little as possible.

How it is Done

By keeping a record of the previously calculated bounding box (i.e., the one for the previous frame), it is possible to calculate the overlap area between boxes. By iterating through all the detected blobs found in the previous pipeline stage and calculating the overlap between these and the previous frame's bounding box, the closest ROIs can be identified. The region closest to the previous one can then be assumed to be the one containing the player to track.

5.1.6 Image Stabilization

At this point in the pipeline a close to perfect ROI with the player inside is computed, although the player itself might not be centered perfectly inside the bounding box. This offset is a result of arms or legs stretched out in a motion that is not symmetrical, e.g., kicking the ball. On the other hand, running produces symmetrical distances from the center, with one foot in front and the other out back. Therefore, an analysis of the foreground mask is made to center the player.

How it is Used

Examining the player shape and then deciding where the center of the torso is enables the player to be perfectly tracked regardless of outer motion of the limbs. The problem is then how the center can be located, which in our case cannot be done with video image (caused by a lack of frame resolution). However, the silhouette of the player can be used and the torso is found by computing the center of mass (i.e., the foreground mask with the centroid weighted) of the silhouette. This allows the player's torso to be consistently located in the center of a frame, with the ROI expanding or subtracting depending on the action.

How it is Done

In order to take into account cases where limbs extends from the center torso, the mean foreground mask is computed. The placement where the mean of the foreground is should more or less always be the center of the body, because the torso is the most significant region of the foreground. Moving the ROI center to the mean foreground region results in an near perfect center. One could extend this approach by dividing the image into smaller tiles, say three-by-three inside the ROI. The sum of foreground elements within each region is computed and the one with most weight (largest sum) is the center. Fortunately, only using the mean calculation is sufficient to obtain a proper center that can be used to create the final image.

5.1.7 Image Creation

Three components are now computed: the foreground mask, the normalized image and region of interest. The second to last stage is to take these components and create a new image that is to be used with the calculation of optical flow.

How it is Used

While each of the individual stages in the pipeline work over the same original frame, they each consume and produce different data types. Particularly, each stage either filters out or narrows down noise elements and the ROI respectively. The task then boils down into using the noise-free masks and smallest ROI to extract a portion of the original video frame to get a centered soccer player sequence. The way this differentiates from the previous stage, image stabilization, is the fact that image stabilization only computes the center and nothing more. As a result, an additional creation stage is required.

How it is Done

Images are created by first converting the original frame to grayscale and then copying the grayscale pixels from the frame defined by the foreground mask. Any foreground elements within the stabilized image but outside the region of interest is ignored. The location in which the player-pixels are copied to are specified by the image stabilization stage and the result is a centralized image of one player in grayscale, exported as a frame sequence (a set of images to create a sequence, rather than frames to create a video sequence).

5.1.8 Size Normalization

Although the image is now stabilized to the center, it is still lacks consistency in size, which needs to be normalized.

How it is Used

This requirement come from the fact that soccer player sizes varies across the field, caused by the distance from the camera. If the size difference between two sequences of the same class are to be classified, it might not correctly place the query sequence because the size makes the optical flow descriptors too different for the sequences to appear the same. Becayse of this, every frame sequence is normalized according to the median height of the player within a sequence.

How it is Done

To normalize the size, the median player height over all images within a sequence is computed. This factor is then applied to every image in the sequence to zoom in on the center, essentially up-scaling the player without changing the image size. This makes the image consistent across sequences and Figure 5.2 on page 35 illustrate the final result. Additionally, to prevent up-scaling the image too much and remove as much as possible of empty ROI areas, we resize to only half of the ROI and crop out the remainder. The resulting ROI is then 100×100 pixels in size, but contains the same amount of information as the original ZXY ROI size of 200×200. As a result of changing the size of the final ROI we also reduces the computational time required when performing frame-to-frame similarity measuring in Chapter 7.

5.2 Results

It is important to have a proper tracking of the players, because a poor implementation here would lead to inaccurate results in the optical flow and later on during the classification. Additionaly, the proposed tracking algorithm relies heavily on the background subtraction being correct, which means that it must be as accurate as possible. We therefore need to test the accuracy on both of these two results, but objectively asserting the quality of image processing is a potentially challenging problem. Research by Wang et al. [55] briefly covers the different methods of evaluating the quality, but also why automatic testing is difficult. The primary reason for the difficulty is that common algorithm comparisons work primarily on color intensities, but do not include structure in a sufficient degree. Structure is especially important when tracking individual objects, as without structure it is not possible to tell which part of an image is the objects of interest. In essence, verifying object tracking can only be done manually with experts, which itself can be costly, but more importantly, subjective.

In order to objectively verify the tracker results, two assessments are made: The first one is a manual inspection of the background subtraction with margins and the second one is a comparison of the extracted bounding box (tracking result) compared[1] to a manual plotting made in the `Online Training Tool`.

5.2.1 Background Subtraction

To objectively assert the background subtraction algorithm we set up two criteria to define a pass or fail label to the computed mask. The other option would be to manually compare every single pixel and foreground mask, which is a tasked deemed unfeasible. The criteria are:

1. Only players should be foreground
2. Player foreground should be whole

The definition of a *whole* player is a connected foreground mask that encompasses the entire player (i.e., one blob covers the player entirely) without cutting out the head, arms or similar and not extending past the player.

Several iterations over the different parameters available to the MOG2 algorithm were attempted before the optimal result was obtained. For every iteration, the aim was to improve the two criteria, and was done mostly out

[1]Comparing results to an expert on the field is commonly referred to as Gold Standard Testing, see Section 6.4.3

of trial and error. Considering the image of the player alone, the background subtraction algorithm fulfills both criteria (cf. Figure 5.2), but not always.

Usually, a couple of frames are required by the MOG2 algorithm to learn the proper definition between foreground and background, resulting in a failure of the first criteria. But, with sequences spanning several seconds, this is considered an acceptable error. On the positive side though, the second criteria is almost always correct with only a single observed instances where the player is broken into two foreground blobs. In the end, visual inspection of background subtraction is encouraging, as being able to correctly identify foreground elements most of the time.

The background subtraction is only part of the tracker algorithm though, with the actual desired tracking being a ROI of a soccer player. Fortunately, determining the accuracy of the ROI is much easier than verifying the foreground mask and can be done using an overlap test.

5.2.2 Bounding Box Overlap

To re-wrap the requirement for a successful tracking of a player, a bounding box is to be found that is no larger or smaller than the player in question with the player perfectly placed in the center. Centering of the player depends completely on the placement of the ROI and is therefore not necessary to check the quality of the centering directly. To properly verify the computed ROI, a selection of sequences from three different actions are manually annotated with bounding boxes around the player, which is then compared against the tracking algorithm's bounding boxes and calculate the percentage overlap where the result is expressed as the percentage of overlap (which abstracts away the changing players sizes and required bounding boxes).

The overlap computation is described in Code Snippet 5.1 which, in short, computes the intersection between the tracked and annotated bounding boxes. If the tracking box is larger than the annotated, a percentage above 100 is given. If larger tracked bounding box is computed, but it does not encompass the entire annotated bounding box, then the coverage is computed from the overlap area alone. The Code Snippet 5.1 includes the calculations required and uses OpenCV's `Point2f` class, which contains the two variables `x` and `y` that represents a position in 2D space.

Code Snippet 5.1: Function for calculating bounding box overlap using OpenCV's Point2f class

```
float boxPercentageOverlap(Point2f start_expert, Point2f end_expert,
                           Point2f start_tracker, Point2f end_tracker)
{
    if (fmaxf(start_expert.x, start_tracker.x)
           > fminf(end_expert.x, end_tracker.x)
        || fmaxf(start_expert.y, start_tracker.y)
           > fminf(end_expert.y, end_tracker.y))
        return 0;

    Point2f start_intersect
        = Point2f(fmaxf(start_expert.x, start_tracker.x),
                  fmaxf(start_expert.y, start_tracker.y));
    Point2f end_intersect
        = Point2f(fminf(end_expert.x, end_tracker.x),
                  fminf(end_expert.y, end_tracker.y));

    float areal_expert = (end_expert.x - start_expert.x)
                       * (end_expert.y - start_expert.y);
    float areal_tracker = (end_tracker.x - start_tracker.x)
                        * (end_tracker.y - start_tracker.y);
    float areal_intersect = (end_intersect.x - start_intersect.x)
                          * (end_intersect.y - start_intersect.y);

    if (areal_intersect == areal_expert)
        areal_intersect += areal_tracker - areal_intersect;

    return (areal_intersect * 100.0f) / areal_expert;
}
```

Measurements are taken with six sequences of the actions "run", "side-jump" and "kick" for sequence S5, S20 and S21 respectively[2]. The tracking algorithm is set to operate with a small margin, to account for small errors from MOG2, which means that each sequence is run with both 0 and 6 pixels margins. The median results are given in Figure 5.4 for all three sequences, labeled with sequence number first and an *M* if 6 pixels of margin is used. It also contains maximum and minimum error bars for the smallest and largest percentage overlap found within the sequence.

First of all, the pixel margin of 6 pixels is a necessity to ensure sufficient coverage, as discovered during extensive manual testing and verification, and as observed in Figure 5.4. While a small bounding box as possible is desired, having it too large is better than too small, as it prevents a soccer player from being cropped. With two out of three sequences' average overlap coming close to the 100% mark it makes the tracking a success, with a few remarks.

The most important remark is that the annotated video sequences might not be properly set, that is, correctly setting the bounding box manually is still a subjective task prone to under- or over-provisioning of the box. Secondly, the

[2]Taken from the classification skeleton transfer sequences, see Section 6.7

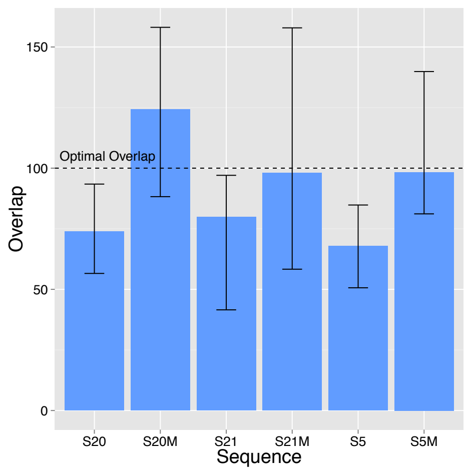


Figure 5.4: Median percentage overlap of bounding boxes compared to expert, plotted with maximum and minimum bars. The optimal overlap is 100%, marked with a dashed line.

added margin to the player tracking algorithm is slightly larger than required, but it is set to ensure no player cropping occurs regardless of the situation, resulting in larger than 100% overlaps. If examined more closely though, this massive overlap is actually not more than a few pixels (as shown in Figure 5.5), which is a tolerable difference.

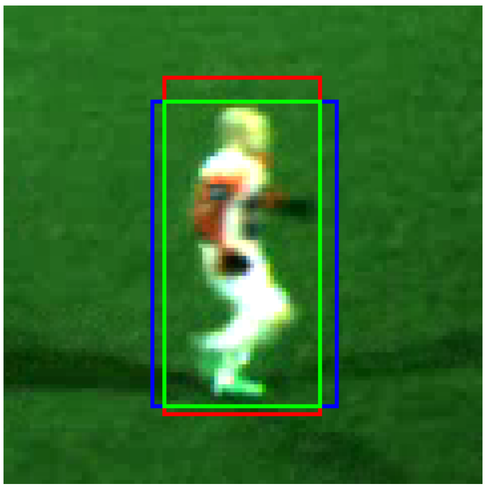
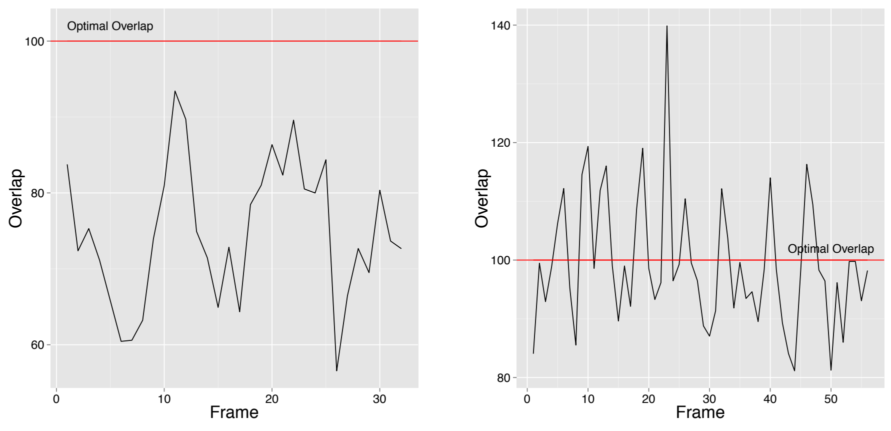


Figure 5.5: Comparison between expert plotting and the tracker algorithm, where the red bounding box represents the tracker, blue the expert and green is the intersection between red and blue. Optimal tracking should consists of only the green rectangle.

Looking closer at two of the sequences, S5M and S20 in Figure 5.6b and Figure 5.6a respectively, another couple of observations can be made: Sequence 5, with 6 pixels margin, has the most common action performed during a game (running). Yet, it too consists of a high fluctuation on the overlap coverage of about ±20 percentage points compared to the 100% mark. Although, checking each frame's bounding box individually with the tracking data results, we actually consider it to be better than the manually annotated bounding box, possibly hinting at inconsistencies from the annotator. Perhaps more important is the fact that even a single pixel difference between the tracking ROI and annotated ROI will cause large percentage differences. A better approach to measure the difference could be to examine the number of pixels the two bounding boxes deviates from one another, but that in itself does not describe the player coverage differences, which is what we are interested in. For example, 1 pixel offset could mean both over and under 100% coverage, unless signed numbers are used in the offset. Even then there is a problem of combining the difference for all four edges into one number that describes the offset between the two ROIs.



(a) Sequence S20 (b) Sequence S5M

Figure 5.6: Percentage of bounding box overlap of the tracker compared to an expert, with the red line indicating 100 percent overlap.

The other sequence, S20, has one single data entry that shows what happens to the player tracking algorithm when background subtraction fails to produce a proper foreground mask, causing the dips in Figure 5.6a. This drop occurs both at sequence start and during the actual kicking of the ball and is considered less than optimal tracking results.

However, the accuracy and consistency of the tracking results rely on the background subtraction, which has already been approved. The same applies here: While the player tracking algorithm has some fluctuations and failures in some special cases, it still manages to produce acceptable ROIs in almost every single image and sufficiently accurate for our proposed solution. Moreover, having a bounding box slighly larger compared to an expert's selection is better, as having it too small might cut out player limbs, which is not desirable.

5.3 Remaining Issues

The initial results are promising for use with selective video sequences where tracking can be guarantied to always work, but there exists cases where the current tracking implementation fails to properly track a player. This section covers the cases that causes tracking failure and presents potential improvements for future works that are required for a fully functional system. The list provided here is not necessarily complete, but provides the most frequently observed problems and gives a brief explanation of why they occur, starting with the most common behavior: overlapping players.

Overlapping players

The problem of overlapping players occurs when they either interact with one another or the players are standing in a line. The background subtraction algorithm correctly segments out both of the players, but the foreground mask has the two player blobs connected as a single blob. Consequently, the ROI will then encompass both of the players, despite only one of them being the player of interest. Additionally, it offsets the bounding box center to the center of both players, an undesirable effect. Interacting players can possibly be separated with an image segmentation stage that cuts the single, large player blobs into two smaller ones.

The problem of overlapping players extends to players crossing paths as well, but a possible solution to this is to attempt every possible tracking when a crossover occurs. For instance, the tracking algorithm could in this case keep a list of all the players participating in the crossover and then attempt to follow a single player from this list. If the tracked player is lost (either by being outside the ZXY ROI or the ROI suddenly jumpts to another position), the system rewinds and attempts to follow another player in the list. Using this technique could result in better tracking of the soccer players, but real-time execution speed and number of possible tracking branches could become too high. Furthermore, this techinque might not work when players cluster together.

Cluster of players

Clustering of players is an extension to the scenario of overlapping players and is a more severe and probably more difficult problem to resolve. It typically occurs at long distances from the camera array, where each player is not large enough to completely occupy the ROI alone. The most notably case are when all the players are defending/attacking at the same spot. If this happens, then the algorithm tracks in a seemingly random fashion, often jumping between players.

Animated advertisements

A separate problem to the background subtraction is the issue of the animated advertisement found around the edges of the field. These moving images alter their pixel intensity and is as a result mapped as foreground. A quick fix would define these pixel points as background, but the players would also be mapped as background when they cover the advertisement strip. An additional segmentation step to separate advertisements and players would then be required.

Obstructions

Anything blocking the line-of-sight between the player and the camera is a potential problem for the tracking system. While it remains to be mostly a camera placement issue, there is still cases that are a real challenge to overcome. Rain, while not blocking the perceived line-of-sight, does add a significant noise to the image and essentially obscures the view required for a perfect tracking. Even worse if it hails, which has an even larger blocking effect that water droplets, which in theory would invalidate any background subtraction algorithm. Although more of an extreme case and can probably be disregarded, it is still an issue considering the fact that the Bagadus system operates with outside stadiums.

Background Subtraction

Both the strength and weakness of the tracking algorithm is the background subtraction. While it is durable enough to compute correct foreground masks with the current installment of the Bagadus system, it must still be tuned for optimal results. If lightning conditions, player outfit or weather type changes, it can cause tracking failures. As an example, the background subtraction configuration used in this thesis manages to map the white and red players with high accuracy, but the opposing team is frequently mapped as background. Even worse is the case of players standing completely still, as they slowly fades into background, caused by the learning history found in the MOG2 algorithm. When a player is faded into the background, the tracking algorithm either stops working or jumps to track another player. To solve these problems, either a multitude of background subtaction algorithms must be used or a better configuration for the parameters to MOG2 must be applied.

Low pixel density

Even with a perfect configuration for MOG2 background subtraction algorithm, there are still problems with ambiguity caused by video resolution, noise and compression. The video sources used in this thesis have a relatively low resolution, considering the 50 to 150 pixels player heights, to have the foreground mask successfully computed to all the corners of the field. While strictly speaking not an issue, a camera sensor with higher resolution could potentially decrease the amount of ambiguity for background subtraction and increase the detail in players to allow for player separation in clusters and overlaps. In short, a higher resolution camera can reduce, remove or at least aid in resolving the issues mentioned in this section.

5.4 Summary

Player tracking is part of the content-based video retrieval extraction process, where imaged soccer players are extracted and exported as small video sequences. Specifically, the tracking algorithm is a pipeline that uses ZXY position data and a background subtraction to find, isolate and center soccer players within a small frame. By taking noise and small margins of errors into account, the tracking algorithm can produce results quite similar to a human's perception of the region of interest a player occupies.

Even though the tracking algorithm operates within acceptable parameters for our prototype, there are still issues that needs to be resolved before a full system can be finalized. These issues mostly concern the precision of the background subtraction, occlusions and noise in the video image and the separation of the players. If these issues are solved, then the tracker can, for any given scenario, successfully identify and center the soccer players found in the Bagadus system.

The tracking algorithm is used for extracting video (video extraction) from a soccer game to obtain small sequences of players. These sequences are then to be annotated in a crowdsourcing platform or classified in the action retrieval and pose estimation algorithms.

Chapter 6

Video Annotation

Content-based video retrieval for pose estimation requires a database of annotated sequences that contains skeletons. The annotations to obtain consists of a set of points that specify where in a frame the joints of a soccer player is, which is used to compose the skeleton, and annotating the action class. The annotation must be done manually, because no motion capture data is available. Annotating hundreds of images using hired experts, people who annotate every skeleton without any errors, can be costly and doing it ourselves would require weeks of effort. We therefore employ a crowdsourcing platform where people around the world are asked to annotate frames for a small fee. The difficulty is then how to obtain accurate data from non-experts (i.e., inaccurate and abberant annotations), which we attempt to resolve by using quality control mechanisms and having each frame annotated several times to then have all the contributions for a single frame merged into a single skeleton. In order to support annotation of soccer players, an `Online Training Tool` *has been developed that can register and store user inputs.*

This chapter contains an explanation on what the crowdsourcing platform is and what crowdsourcing is generally used for. Then, the datapoints to be annotated in the `Online Training Tool` *program (see Section 4.1.2) is presented. With the data fields defined, a set of design iterations are made over the* `Online Training Tool` *to maximize system usability and reliability, as well as defining filtering and validation functions to get the best skeleton data from the user annotations.*

6.1 Crowdsourcing

6.1.1 Terminology

We use the terminology for crowdsourcing as presented in [8] by Brabham and this is used for the remainder of the thesis. Brabham defines crowdsourcing as using a large group of people (a crowd), connected over the internet to collectively solve problems ranging from coming up with ideas to producing content. People, or *crowdworkers* (*workers* for short), participate with their different skills and knowledge to usually solve small individual *tasks*, where a single task is later combined into a larger solution. Furthermore, these tasks can be provided by anyone from big companies to hobbyists and private persons. The direct opposite of crowdsourcing is to hire an *expert*, a paid worker that possesses all information and skills to provide perfect answers to any task in a particular field, which can be slow and extremely costly.

Structuring workers and tasks is done using a platform. A platform is essentially just a website that provides functionality to hire and pay registered workers and starting *campaigns*. A campaign can consists of any tasks, but the tasks should be as small as possible, for workers to actually be willing to solve it and as individual as possible, i.e., each task is independent of other tasks to achieve maximum scaling. Additional safeguards and quality controls are also a necessity for campaigns, because workers may attempt to cheat the system to earn money or simply provide low quality work.

The crowdsourcing platform used in this thesis is *Microworkers* [33]. It is one of many platforms to connect employers (task providers) with workers. An employer simply needs to create a new campaign and specify the necessities, like an instruction manual and the time each user is expected to use on the task to finish. Microworkers also provides task result ratings, but we need to verify user productiveness in our online training application according to our own requirements. Alternatives to Microworkers exists, for example Amazon Mechanical Turk, a rather successful crowdsourcing platform used in many different research projects.

6.1.2 Related Work

Crowdsourcing platforms have had an increase of use in everything from scientific research, text translation to creative work [22]. Particularly, scientific research has access to a large user base of people, without having to manage worker registration and payment. Neither do they have to advertise for the research in order to get users to sign up in the first place, as everything is already provided by crowdsourcing platforms, e.g., Amazon's Mechanical Turk.

Using Amazon's Mechanical Turk, Loni et al. [31] attempted to use crowdworkers to aid in the labeling (or titling) of images, a quite common task to use with crowdsourcing. Additionally, workers were also tasked with verifying other workers' labels to provide quality controls. With this technique, Loni et al. were successful at increasing the accuracy of the labels compared to only using the crowdworkers for labeling.

Another interesting problem with crowdsourcing is whether they can be used to solve more complex tasks and still perform to the same level as experts. Hsueh et al. [23] investigated this by having the workers group text segments as positive or negative for a politician, a considerably more challenging problem than defining a title for an image's content. By properly filtering the crowdsourcing data, the authors were able to achieve reasonable accuracy, showing that crowdsourcing can be used to solve complex tasks.

Even more interesting is the use of crowdsourcing to solve tasks that are not a simple selection between good or poor quality, but requires direct interaction for the crowdworkers. Su et al. [51] used a crowdsourcing platform to obtain bounding boxes around objects. With their near-perfect results, and our similar problem of obtaining player skeleton positions, makes the crowdsourcing platform a viable option for this thesis to obtain joint positions for soccer players.

6.2 The Task

The task each worker is assigned to consists of entering and annotating a set of points that define player joints, a bounding box that defines the player ROI and the action class. The annotation tasks operates with individual images obtained by exporting frames from video. Consequently, a frame and an image are used interchangeably for the remainder of this chapter, with both describing a single frame of a video sequence that is to be annotated with a skeleton and action label.

Optionally, instead of providing a full-length video clip (e.g., a soccer game), a set of selected sequences can be entered into the annotation tool. If the action class of these sequences are known, then it is possible to configure the tool to only require the skeleton annotation and bounding box selection. Action class selection and player count inside the bounding box is then disabled. In the case that an entire game should be trained, the functionality should be activated.

6.2.1 Source Frames

All the video footage of soccer players used for annotation comes from an earlier archived game from the Bagadus system, available online at [47]. Using the mapping technique described in Section 2.3, we can obtain a small Region of Interest (ROI) for each player and export this region as an image that is uploaded to a web server with a unique identification number. A website is then provided with a user-interface to support entering of data elements. Workers are then asked to annotate soccer players using this tool, with each worker assigned a random frame. Randomization is done for two reasons: The first is to provide variation for workers, as annotating the same second can be too repetitive. Secondly, by evenly distributing high and low quality work across all frames should provide a decent overall average quality. This process is done for every frame of every player for all video sequences, resulting in tens of thousands of annotated skeletons.

6.2.2 Bounding Box

When a worker is presented with an image to annotate, they have the option to select a bounding box (ROI) around a soccer player. If a worker do select a bounding box, then the selection is enlarged to assist the worker in annotating the joint locations. Since bounding boxes is not a primary source for the skeleton, but merely an aid to the workers, it is not kept when merging the results. Furthermore, the workers must also define the number of players present in the bounding box or full frame if no ROI is selected. This is done to be able to filter out any sequences with more than one player in the ROI, but is never utilized in this thesis and can be added to the merging of the results if desired.

6.2.3 Skeleton Plotting

After an optional bounding box is selected, the worker is to annotate a skeleton for the soccer player, consisting of 13 joint locations: one for each head, shoulder, elbow, hand, hip, knee and feet, covering both left- and right-side limbs. Typically, more than 13 joint locations are used e.g., [10, 19, 59], but 13 allows for accurate placement of all limbs given the resolution of the source frames (e.g., annotating the neck would be unfeasible). In the rest of this chapter, a worker's annotation for a single joint position is referred to as a *click-point*.

6.2.4 Motion Labels

Before submitting the annotated skeleton, the worker must define the action being performed from a set of predefined classes. Determining the action from a single frame is impractical and a short video sequence of the frames before and after the current frame is presented to the user. The server already stores these frames which are then used to create the video clip.

6.3 Design Considerations

One of the biggest concerns when using a crowdsourcing platform is the large variety of the individual workers' background and expertise. It is therefore important to have the task clearly defined, but also have a system that handles the many types of answers (e.g., everything from correct ones to hackers) elegantly to ensure maximum reliability. Additionally, with an almost infinite number of possible click-points in the task, ensuring precision and consistency in the dataset can become a serious problem.

Schulze et al. [44] presents five quality assurance mechanisms for use with crowdsourcing: Qualification Test, Qualification Restriction, Gold Standard Test, Majority Voting and Validating Review. Two of these cannot be applied to our system, namely Qualification Test and Validating Review, as a qualification test might discard too many workers and validating review is not feasible with the amount of distributed tasks. The remaining three mechanisms are discussed in the design iteration they are implemented in.

To ensure both system reliability and user friendliness, a set of design iterations were made. The goal with each of these iterations are to ensure support for a full-scale crowdworker campaign, but also to address any observed issues. To this end, the design iterations are directly linked with pilots: test campaigns set out to iterate over designs using real-world data and feedback. The following sections contains the three pilots used, with each one describing the goal of the pilot, any new algorithms or issues resolved and initial results from the design iteration, resulting in the final `Online Training Tool` program shown in Figure 6.1.

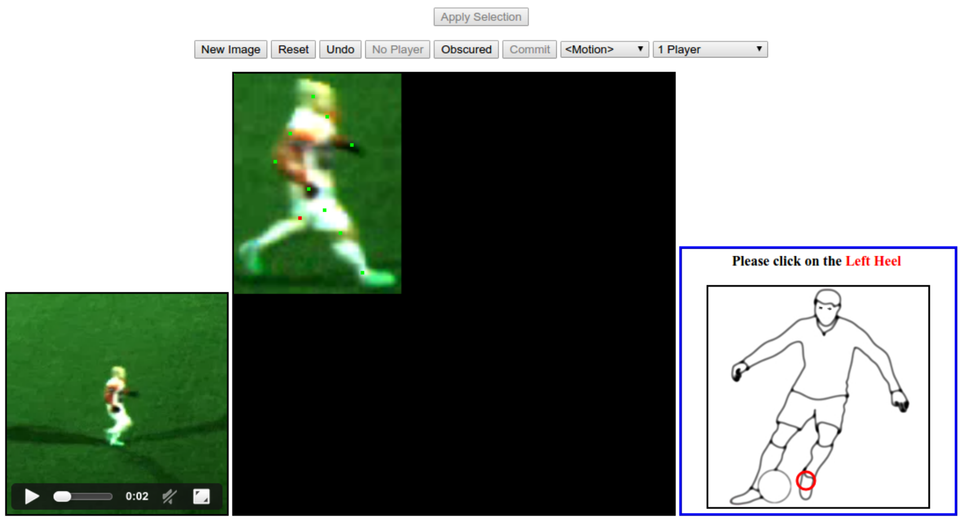


Figure 6.1: Final version of the `Online Training Tool` with all controls enabled. From left to right is the action preview window, the annotation window and the annotation request window. Note that naming of the joints are different to better define where to click for the crowdworkers.

6.4 Design Iterations - Pilot 1

The first pilot was primarily a test to see if it was even possible to give a point-and-click scenario of skeleton annotation to a non-expert and still obtain usable results. However, before the pilot could be run, a set of precautions and quality systems were developed into the `Online Training Tool`. The test sequence used in this pilot consists of a single player performing the action "run" without any other players visible in the ROI and has a length of 113 frames.

6.4.1 Result Filtering

Obtaining the data is only one part of the entire process of using a crowdsourcing platform to gather data. Before the first pilot was run, a set of filters and functions were implemented to both validate results and protect the system from malicious workers. Filtering of the results that do not conform to a desired accuracy or correctness keeps the resulting data set accurate and reliably [23]. Moreover, considering that crowdsourcing can be a paid labor, as in this thesis, there are cases where a worker attempts to maximize profit while minimizing the work required, e.g., by entering random data. Some workers may even go so far as to attempt to break down the system completely rather than performing any useful work.

The system requires protection against the two types of malicious workers, namely *cheaters* and *system breakers*, as well as being able to identify work that do not meet the desired quality. In the case for this thesis, the quality measure is how accurate and correct the placement of every click-point are within an image compared to an expert's. Section 6.4.2 describes how the system is design to protect against system breakers and Section 6.4.3 describes how to filter out cheaters and validate results.

6.4.2 SQL Injection Protection

SQL injection occurs when a user actively aims at altering a query to either delete or obtain information in a database. Since the data used in this campaign does not store any sensitive or personal information, it makes information gathering attacks less of a concern. Deletion, on the other hand, becomes extremely important. Because every worker is paid, loosing any of the entered data costs both time and money to reproduce. A couple of operations must therefore be taken before any database-related operation occurs.

How it is Used

The first step is to provide input verification: No input can never be trusted in online applications, be it with URL arguments, cookies or HTML forms. The implementation in this thesis utilizes cookies to transfer information between the host and the server and makes this the primary source for SQL injections. As a result, every cookie key/value pair must be checked for both its existence, to make sure crashes do not occur with missing values, and that it has the correct data type input. Any worker who attempts to break the system in any way will have its work effort rejected and has its entries in the database removed.

How it is Done

In short, every data entry point is type-checked and verified to ensure the values are within expected values. For example, verifying integers could involve checking the signedness of the number, and for strings, escape special characters and prevent embedded SQL statements in the string from executing. PHP, the server-side language used in this thesis, have *prepared* sentences that can be used for type-checking and automatic string character escaping, but requires a lot of code. Therefore, a library called *MeecroDB* is used to simplify query type-checking and perform input validation in PHP. With SQL injection protections, malicious workers are unable to break our system.

6.4.3 Gold Standard Testing

Rejecting users who do not conform to the desired accuracy can be done by comparing a worker's click-points against a ground truth, set by an expert, and measuring the distance between the expert and worker. Furthermore, it keeps the resulting dataset reliable, because no click-point can be farther away from the ground truth that what is defined by the threshold. The process of performing ground truth tests is also called *Gold Standard Testing* and is one of the mechanisms presented by Schulze et al. [44].

How it is Used

Throughout a task, a worker is at random times assigned an image that is annotated by an expert, i.e., gold standard test. There are no differences in the training tool between a gold standard test and normal annotation and appears as a normal annotation request. This ensures consistency in both the results and gold standard tests with the worker performing the same quality of work for all images. On the system side, however, the annotations made for gold standards are instead measured and entered into a separate list as accepted/rejected entries. This allows for real-time validation of worker accuracy and is used to accept or reject contributions in the Microworkers platform on task completion.

The gold standard testing mechanism is not completely perfect though, as only a handful of frames are checked which is then used to create a verdict on a worker's performance. Distributing more ground truth tests during a task would definitively increase the baseline for making a worker quality assessment, but it would also decrease the amount of productive work (i.e., annotations for new images) as it is desirable to have the images annotated per task relatively low, e.g., around 15 to 30. In despite of this shortcoming, gold standard testing proved sufficient, as the first pilot displayed a consistently high or low quality effort for all the frames a worker annotated, with the quality measured in distance from ground truth. Additionally, all workers attempting to perform the task quickly, i.e., to earn as much money as possible, will end up failing all the gold tests and not get paid.

How it is Done

The test against ground truth is done by measuring the euclidean distance between worker input and ground truth on a per-joint basis. Whenever a single joint location is farther away than a specified threshold from the ground truth, a golden test failure is logged in the system. A worker who do not pass a certain amount of gold tests then gets its work effort rejected, meaning, the annotated work is not included in the final result dataset and the worker does not get paid.

Unfortunately, it is not enough to provide e.g., two gold standard tests at the beginning of each task. The reason being that a user can become more tired at the end, and therefore do far worse than at the beginning, or the other way around and perform better. A more important argument for having a proper distribution of gold tests is that workers can discover which images of a task are the gold standards and then only do these properly while solving the remainder of tasks as fast as possible. Microworker's workers also comments on the tasks, which makes the revival of the gold standard tests even more likely. However, by randomizing when a ground truth sample is to be tasked to a worker, and have a large enough ground truth set to not have too many repetitions, greatly decrease the chances of finding these tests.

The function that determines whether a task should be a gold sample or not cannot be a pure randomization, because random selection could in theory make a worker never have to do a ground truth test image or only having to do them. Additional constraints that ensure an approximately, even random distribution is therefore preferred and the gold samples should not be distributed to a worker more than the minimum required gold samples, because it reduces the amount of new skeleton annotations.

Code Listing 6.1 implements the PHP function to determine if a gold sample is to be tasked to a worker. It uses two global configuration variables `training_max` and `control_times` that defines the total number of training images per worker and the number of gold samples to be distributed per training set respectively. The variable `control` gives the number of gold samples already processed and `index` is the number of images (non-gold samples) that have been annotated. The function randomizes when golden standard tests are administered, but also has a distribution guarantee. This guarantee ensures at least every $training\_max/control\_times$ image is a ground truth test with no more than $control\_times$ ground truth tests per task.

Code Snippet 6.1: Function to determine if a Golden Standard Test is to be given as a task.

```
function shouldBeGoldSample() {
    global $training_max;
    global $control_times;
    $control = $_SESSION['control_count'];
    $index = $_COOKIE['training_idx'] + 1;

    if ($control >= $control_times || $control_times <= 0) {
      return false;
    }

    $interval = intval($training_max / $control_times);
    $expected = intval(ceil($index / $interval));

    if ($control < $expected && $index % $interval == 0) {
        return true;
    }

    return rand() % $training_max == 0;
}
```

6.4.4 Initial Results

Although the first pilot proved that crowdsourcing can be used for annotation of soccer skeletons and that the user-interface, input checking and validating operates as expected, it still has several shortcomings that needed to be addressed. The most significant of those is the lack of sufficient logging of non-position elements, like gold sample click-points. Because gold sample log entries are computed on the server and then updates a counter for how many failures a worker has, it is impossible to derive the main reason for any failure. With the fact that every user who completed a task failed all the gold samples truly underlines the need for logging gold standard tests' click-points. Although, a theory for these failures may come from the acceptance threshold being far too strict, a theory supported by a new implementation of the gold standard tests.

Instead of measuring the individual click-points' distance from ground truth, the mean of click-point distances are computed and an acceptance deviation threshold is set to 25. 25 pixels is chosen to allow one or two far-off joint annotations while still maintaining a high quality for the skeletons. This new implementation resulted in about half the tasks being accepted, which is equivalent to the number of accepted tasks determined by an expert. With all the issues resolved, a new pilot was submitted to the Microworkers platform.

6.5 Design Iterations - Pilot 2

The second pilot's key issues were to address the logging of user activity and test how many images a worker is willing to annotate. Because the logging requirements were designed and implemented before Pilot 2, only a verification was needed to test if it operates correctly. Additionally, users were encouraged to guess the location of partially obscured limbs in an attempt to reduce the number of joints marked as obscured. To test the willingness of workers to annotate soccer player skeletons, every task was configured to have 40 images and 8 gold standard tests, with the option to stop the task halfway, i.e., after 20 images and 4 gold samples. All the images and gold standard tests are from the same sequence used in the first pilot, i.e., a video sequence of a single player running from left to right across the field.

A new worker task validation process was also implemented, which utilizes majority voting to further improve the filtering and to address the issues present with pure gold standard testing. As observed in Pilot 1, the gold standard proved to be too restrictive with its per-pixel offset and small sequence coverage, but is included in this pilot as well to compare it against majority voting.

6.5.1 Majority Voting

Majority voting is the second quality control mechanism presented by Schulze et al. [44]. Essentially, the workers validate themselves, rather than by an expert, by only accepting the answer that at least half the workers agree on. This relies heavily on the crowd having a consensus, which is present in the scenario found in this thesis (e.g., everyone agrees that a "leg" is a "leg" and not an "arm"). If the understanding of the body was subjective, then relying on majority voting for result filtering might not have worked.

How it is Used

Majority voting uses the data provided by the workers to identify inaccurate click-points. Specifically, the correct joint location is the one pixel in an image the majority of the workers click on. Any click-points not having the same position as the majority is then considered incorrect. Unfortunately, joint locations cannot be determined solely on pixel position, because even the most identifiable joints locations' can have small pixel differences from one worker to another. Instead, the center of all the click-points is used as the final, correct placement and any click-points that are not within a certain limit are rejected.

Comparing the majority vote and the improved gold standard testing mechanisms resulted in the two performing within the margin of error of one another, indicating that majority voting can be successfully applied to a relatively ambiguous scenario of skeleton annotation. Consequently, and because majority vote filtering is faster, cheaper and easier to use, made it the filtering mechanism of choice for performing task validation on the crowdworkers.

How it is Done

First, the centroid of all the click locations is calculated and a mean deviation is found by computing the mean distance of all click-points to this centroid. Every click-point placed outside this distance would then be considered unacceptable.

Calculating the centroid for use with majority voting assumes that only a single cluster is present in a frame, to which we define a cluster as a set of click-points grouped together with a close proximity, but when there exists more than one cluster of click-points a problem arises (see Figure 6.2). Typically, two clusters occur when the workers disagree of which joint is the left and right and is resolved by adding the previous centroid's location as a click-point to skew the centroid closer to either of the clusters. This does not necessarily resolve the disagreement between left and right entirely, but prevents any in the middle positions from occurring in the dataset.

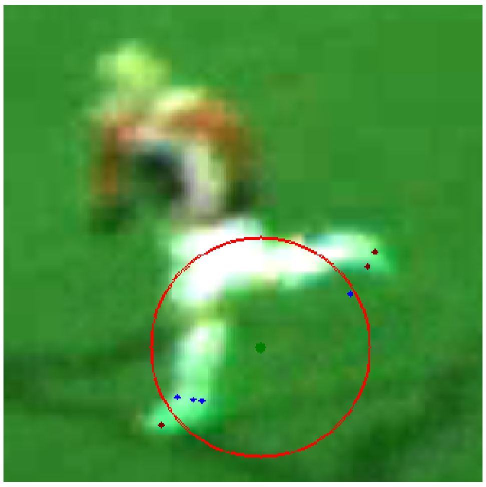


Figure 6.2: Majority Vote Filtering for two clusters. The green point is the centroid of all click-points and an acceptance radius is marked as a red circle, calculated from the mean distance from the centroid to all other points. Blue points are accepted click-points and red rejected click-points.

In an attempt to improve the majority voting algorithm even further, a weight was added to every click-point when computing the centroid. The weight is calculated as the inverse distance from the centroid, which must be computed once before the majority vote filtering can be applied. It slightly improved the accuracy when clusters occurred, but the average accuracy of all the points decreased. Unfortunately, it did not solve the left-right ambiguity either, and is therefore not included in the final algorithm.

To better deal with more than one cluster occurring, and improve the final joint position's accuracy, several iterations with the majority vote filtering are run. For each iteration, inaccurate click-points are removed and the iterations are run until convergence (i.e., the center moves with less than a pixel after each iteration), shown in Figure 6.3.

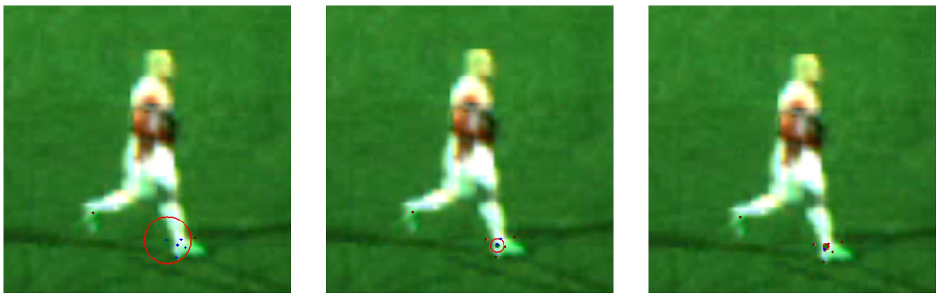

(a) Iteration 0 (b) Iteration 1 (c) Iteration 2

Figure 6.3: Majority Vote filtering with three iterations before obtaining the final position. The green point is the centroid of all click-points with the acceptance radius marked as a red circle found as mean distance from center to all other points. Blue points are accepted click-points and red rejected click-points. Orange is previous iteration's centroid.

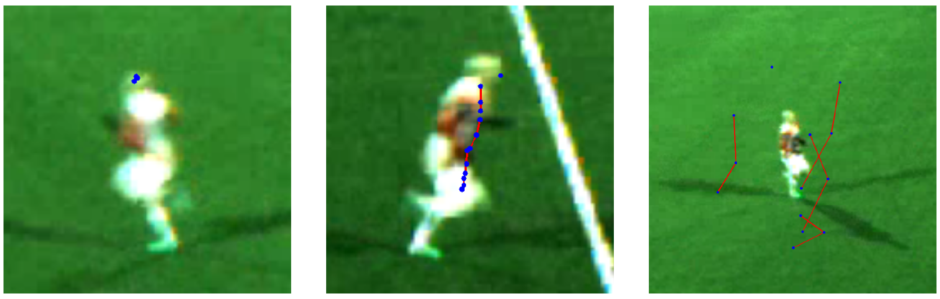
6.5.2 Qualification Restrictions

One challenge with majority voting is bad data that is so far off the actual joint location that it negatively affects the result of the correct plotting, with a set of examples provided in Figure 6.4. While majority voting will in most cases handle these inaccurate annotations just fine, it is still desirable to remove these cases from the set of click-points for a joint before running the majority filter. Another reason for this is to remove bad worker contributions right away, rather than having them present in the merged result. Because of this, we add *Qualification Restrictions*, the last of the quality mechanism presented by Schulze et al. [44], to prevent improper work from entering the result set. Specifically, we seek to remove from our dataset the worker contributions where the entire skeleton for a single frame either forms clusters, lines or is completely random.

Cluster Detection

Cluster detection is used to remove the annotated frames where every point is either in a single location or marked as obscured. These two kind of clusters are annotated by a worker who attempts to click through every single image without actually performing any work. It is faster than creating a line and, fortunately, is quite easy to detect.

By measuring the maximum distance between every joint position, a cluster can be defined as present when the maximum distance between all click-points is less than a given threshold. For the sequences used in the campaign, a four



(a) Cluster (b) Line (c) Random

Figure 6.4: Cases where predefined filters are used to remove noisy or incorrect data. Blue points indicate click-points (joint) and red lines are limbs connecting joints.

pixel radius tested to be a good value. With obscured points stored as the point $(-1,-1)$, it also detects images with all joints marked as obscured. Marking every joint obscured is faster than placing click-points, but not correct (usually at least some body-part is visible).

Line Detection

Another form of speeding through the task is creating lines, but this is much more challenging to detect than clusters. Lines occur when a user clicks across the image, starting from any location, and moving the mouse to the opposite edge. A couple of different approaches can be used for line detection, but they all come with their own drawbacks.

The first approach attempts to draw a line between the points furthers apart from one another. With this line, each click-points' distance from this line is calculated and if every point is less than a given threshold, it is considered a line. An additional benefit of this is that users who only click a couple of joints is also identified as a line, which is desirable. Unfortunately, a line can have a greater curve than the given threshold or a player might be validly annotated with a line skeleton if he is standing upright in portrait. Both of these cases makes the first line detection attempt to strict and potentially harmful to the end result.

The second one attempts to define a left-of or above-of of other click-points. It works by taking every point and check if they are placed in such a way that all other points after the current point, decided by the click-point order specified in the training tool, is either above itself, to the left, right or below. If every click-point have, e.g., all other click-points to the right, it would be a line. Unfortunately, workers do not necessarily create lines in an orderly fashion, and this method was discarded as well.

Because a proper solution to line detection was not found, it is not included in the crowdsourcing filtering process, but left as a future improvement. Problems in detecting lines is both in being able to define a line from a set of points, and not have it be filtered out as a line when a player is standing upright in profile. An additional test to check if every crowdworker annotated lines could be made, and if every worker annotates a line it could be considered correct. However, we rely on the majority vote filtering with iterations to reject the click-points on the line which is too far away the actual joint location.

Detecting Randoms

The last type of speed-through of tasks are workers that are clicking randomly across the entire frame. One could remove these points that lay outside the bounding box, or create a ellipsis region where the most click-points resides. The problem with both methods though, is identifying the ones clicking randomly inside the bounding box. The best way would probably be testing the full skeleton for every frame against a kinematic constraint model (see Section 3.1.3) that would most likely remove random annotations, but creating a kinematic constraint model is outside the scope of this thesis.

Because of this random click-points is not removed by a qualification restriction filter, but let to the majority voting to handle it. Click-points far outside the location for most other click-points causes an additional iteration of the majority filter to be run, but does not affect the final results. It is when the points is placed near, but not correct, that they negatively affects the merged results. Even though they are not removed, it would still be better to filter them out to reduce noise in the aggregated result-set.

6.5.3 Aggregating Results

By now, it is assumed that only good results remain in the dataset, where poor positions are removed by gold standard tests or majority voting. The final filtering step is then to merge the results into a single value that can be used in the classifier. In line with majority voting, the centroid of all the accepted workers' clicked-points are found and exported as the solution [30]. An interesting observation to be made here is that the final centroid location is more accurate[1] than any of the individual users. However, for a proper crowdworker filtering, a sufficiently large number of worker entries per image is required to ensure reliable and precise final merged joint positions.

The same majority voting approach is used for every joint to decide whether it should be annotated as obscured or not. A vote for being visible is represented by a clicked position and if the number of votes for visible is larger or equal to half of the click-points for that particular point, it remains visible. Otherwise, it will be considered not visible, i.e., obscured.

[1]Compared to a ground truth set by an expert.

6.5.4 Initial Results

First of all, encouraging workers to plot joint positions, even though it is not necessarily directly visible, had a positive outcome. The number of obscured points annotated by the workers matched more closely to the expert, a promising improvement. For the logging system, it operated as expected and a particularly useful addition is the worker start and end times that gives the time a worker spent on the task, but also the effort required. It took a surprisingly short amount of time to complete the second pilot: 20 to 40 minutes for the workers who completed the full 40 frames.

Despite the large effort from some workers though, there is still an apparent difficulty with limbs placement as observed in the first pilot. While the legs are mostly, if not always, labelled correctly workers have a tendency to swap the left and right limbs. The annotations for arms does not contain the same amount of accuracy from the workers as the legs does: Clicked values vary between spot on and seemingly random. With proper result aggregation, it is possible to handle the varying degree of accuracy for arms, but it would still be more useful if the click-points were more accurate in the first place. The head, though, seems to be the only location all users tend to agree on, with the difference being smaller than the width of a pixel. All of this information is obtained through the logging of the gold samples as well as the majority voting filtering. As already concluded in Section 6.5.1 majority vote filtering produces the same acceptance percentage as gold standard testing, but is much simpler and faster in use making it the preferred mechanism for worker validation for this thesis.

While the second pilot is also considered successful, a third pilot was run in order to ensure annotating of more than a single sequence is correctly distributed among the workers.

6.6 Design Iterations - Pilot 3

The last pilot was run to ensure the system is stable and support multiple sequences with annotation load-balancing. It was also used to test the annotation limitations of the workers by including a variety of different sequences.

6.6.1 Sequence Load Balancing

To ensure all sequences get tasked to workers and that every sequence gets an equal amount of workers, an algorithm was developed to distribute the workload. This algorithm operates by counting the number of annotated frames within a sequence N and dividing it on the sequence length L, i.e., N/L and produces a workload number, which we refer to as the sequence load number. This number presents the completion rate, and can be multiplied by

100 to obtain percentage. If a sequence is annotated more than once, then the sequence load number will have a number larger than one, describing the number of complete sequence annotations in the digit and partial completion in the decimals.

When a worker logs on an request a task, the sequence with the lowest sequence load number is administered. Also, within this sequence, images are randomly chosen from the least annotated frames. Utilizing this selection algorithm ensures that no sequence starvation occurs (i.e., not worked on due to a bad randomization algorithm) and the workers are evenly distributed across the entire dataset.

6.6.2 Initial Results

All the workload balancing algorithms worked without any issues, but the more interesting part is the result from the three different sequences used. While pilot 1 and 2 used a single, straightforward to annotate video sequence, pilot 3 used three sequences of different types. The first sequence is similar to the one used in the previous pilots, the second is a sequence with several players in the ROI and, the last one a sequence overshadowed by the tribune and is located near the camera array.

The labeling results gave a broader understanding of the different levels of difficulty when annotating frames. Although the set is small, having the workers perform similarly bad or good on the same frames is a good indicator for ease of plotting. Ignoring the usual swapping of left and right labels, the results were best if the image is bright with as little motion blur as possible. Images with higher resolution also makes it easier to annotate, although the overshadowed sequence close to the camera has a lot of random click-points (even compared to the sequence with multiple players in it), despite having more resolution than the other sequences. This indicates that brightness of the video affects the annotation difficulty, but more testing is required to determine if this is the case. In the case of multiple players, there is a consistency issue where the different workers decide to plot different players.

With the different levels of sequence types in mind and no other issues with the `Online Training Tool` present, the full-scale crowdsourcing campaign was set up and run.

6.7 Crowdsourcing Campaign and Results

With all the design iterations complete (i.e., the three pilots), the campaign containing the full dataset was started. Based on the feedback and results from the first three pilots, the task distributed to the workers consisted of 24 images to annotate without any gold standard tests. They did not have to label the action type, as that was predefined from the set of chosen sequences.

For this campaign, a total of 27 sequences (labeled S0 through S26) was chosen from the center camera in the array, with each sequence having the action class predefined. All of these sequences are selected to prevent tracking failure or annotation ambiguity from the workers, resulting in sequences that should be solvable for crowdworkers and possible for the player tracker to follow the player. Table 6.1 lists the sequences, as well as their action label and length.

The final dataset should have each image annotated nine times, although it was ended at approximately five times due to an optimization problem (see Section 8.3.3). Even with a smaller than desired result set, it proved to be sufficient based on the assessments made over the results.

Sequence	Motion	Frames
S0	run	36
S1	run	154
S2	sprint	57
S3	walk-backwards	60
S4	walk-backwards	88
S5	run	56
S6	sprint	49
S7	walk	168
S8	sprint	52
S9	run	115
S10	walk	66
S11	run	48
S12	walk	163
S13	side-jump	47
S14	run	63
S15	run	90
S16	walk	131
S17	side-jump	29
S18	run	44
S19	kick	18
S20	side-jump	32
S21	kick	25
S22	run	54
S23	kick	30
S24	run-backwards	51
S25	run-backwards	46
S26	walk	126

Table 6.1: List of video sequences used in the crowdsourcing campaign.

There are two primary aspects and one secondary aspect to look at when assessing the results of a crowdsourcing campaign for scientific work [58, 60], where the primary aspects consists of accuracy and efficiency, with the secondary aspect on how the crowdworkers' perception of the tasks is. The combination of these factors determines whether using a crowdsourcing platform is better or worse than to hire experts. The first and most important aspect is what we define as *accuracy*, which compares the precision and similarity between the workers and an expert. Secondly, given the accuracy provided by the workers, we evaluate the crowdsourcing platform as a viable alternative to experts, i.e., if crowdsourcing is better in terms of *Cost and Time*. Indirectly affecting accuracy and efficiency results is how the crowdworkers think about the tasks, as an engaging task is more likely to be done better. The crowdworkers perception are presented briefly at the end of this chapter.

6.7.1 Accuracy

Testing the accuracy of the workers is best done by comparing the click-points against an expert's, similar to what is done with gold standard testing in Section 6.4.3, but with a few differences. Firstly, three sequences from the crowdsourcing campaign was chosen instead of the randomly chosen images in the gold standard test:

- Sequence S5 with action "run"
- Sequence S20 with action "side-jump"
- Sequence S21 with action "kick"

Secondly, the filtered and merged set of the workers click-points are used instead of that from individual workers. This should ensure the best annotations the workers can collectively provide and is also more correct, because the joint positions obtained here are the ones actually used in the annotated database for skeleton reprojection.

Figure 6.5 shows the three sequences S5, S20 and S21. The y-axis displays the mean eucludian pixel distance, with maximum and minimum bars. The mean is computed for the non-obscured click-pints in a sequence for a particular limb (which can be seen with *HandLeft* in sequence S20, which is zero because all click-points for this limb are obscured). Additionally, an acceptance threshold is also present to define what would be an acceptable accuracy, with anything above this line being unacceptable. The threshold is set to three pixels, a value determined by inspecting the video sequences' individual images and measure the area under which a click-point can be considered correct, demonstrated in Figure 6.6. Note, however, that this value is only valid for the selected sequences, with other sequences having a lower or higher acceptance radius.

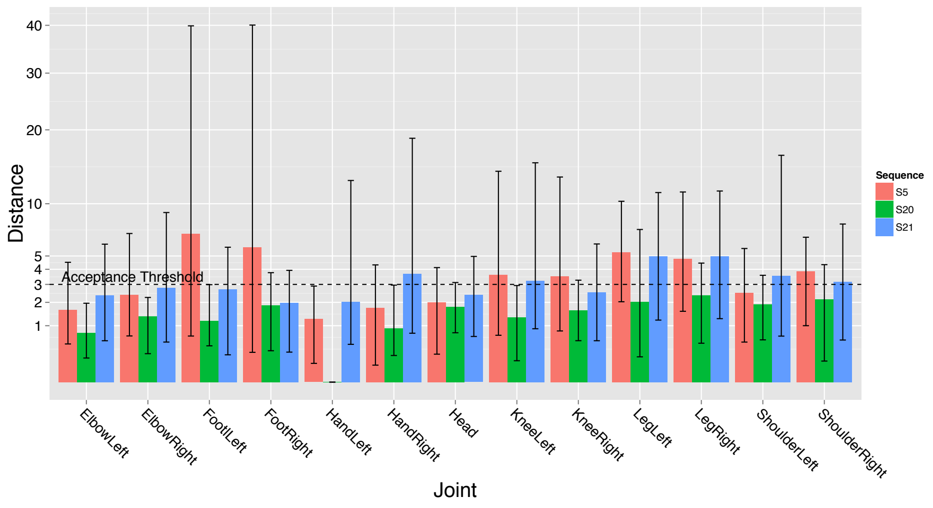


Figure 6.5: Mean difference between the merged crowdsourced joint placements and an expert, measured as pixels, with maximum and minimum bars. Everything below the acceptance threshold is considered acceptable, but is only valid for these sequences.

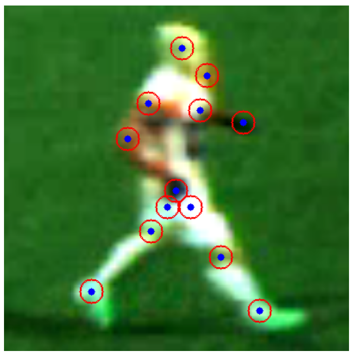


Figure 6.6: Blue joint positions annotated by expert, with red circles showing a 3-pixel margin of tolerable errors for crowdworkers.

Overall, the mean accuracy of the workers is rather good and a lot better than anticipated. Remember that a pixel-perfect match against the expert is impossible, simply because pin-pointing the exact joint position in the up-scaled training images requires no more than a two pixel deviance from the expert before being in a different location. A small variance between the expert and the workers is expected and everything below three pixels is considered correct. The results are also highly dependent on the level difficulty of annotating a player, with "side-jump" (S20) being the simplest as the player is always facing the camera with all limbs clearly visible and indistinguishable.

When limbs are harder to tell apart, the maximum distance increases. The most notable example is S5's feet, which suffers from a mix-up of which one is left and which one is right, causing a large difference between the workers and expert. Actually, most of the maximum error distances measured are from either disagreement between the workers themselves or hard to determine joint locations. For example, in Figure 6.7, the workers disagree on the exact location of the right leg, making the merged joint location more of a random guess rather than an actual estimate. On the other hand, the minimum shows that it is possible to have great accuracy, if not even better than what the expert plotted, as shown in Figure 6.8.

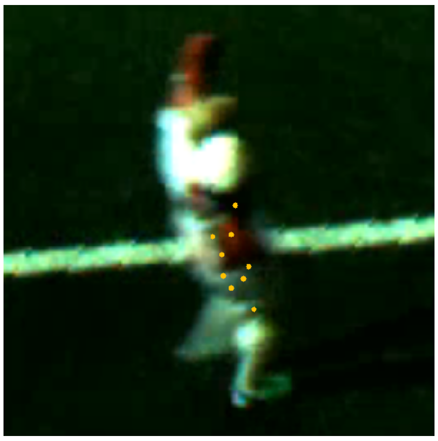


Figure 6.7: Workers attempt at plotting an ambiguous right leg (more precisely the hip) with the individual click-points marked with orange points.

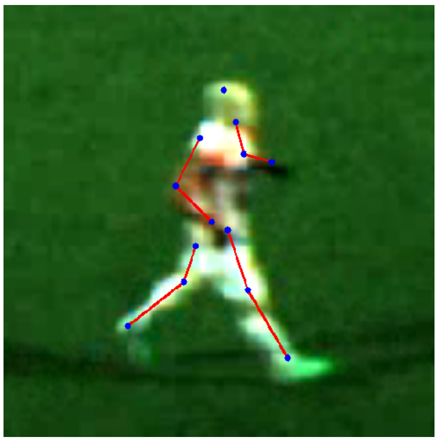


Figure 6.8: Skeleton obtained after Majority Vote Filtering, with blue points marking joints and red lines marking connecting limbs.

A lot of these results depends on the quality of the filtering and merging of individual click-points. As discussed in Section 6.5.1, placing weights on the click-points could probably reduce the maximum distance error on the feet in sequence S5. Adding additional constrains, like a kinematic constraint model (see Section 3.1.3) or make filtering utilize temporal information across frames may further improve the average and maximum distances from the expert. Even though the maximum errors can be reduced, we do not attempt to improve it further, as the mean results are already within tolerable margins.

To summarize: The accuracy of the crowdworkers are rather decent compared to an expert, although, it would be nice to have it even closer to the expert. Most notably is the left-right annotation ambiguity that makes the distances from the experts far too large and would greatly increase the overall accuracy for all joints and sequences if it was not the case. Having more entries per joint per image could possibly improve the results even more, but we were limited to approximately five to seven click-points per image. Yet, with the total mean deviance from expert being 2.66 pixels for the three sequences makes it barely adequate enough to attempt using it in pose estimation and skeleton transfers in Chapter 7.

6.7.2 Efficiency

To determine if a crowdsourcing platform is a viable alternative to experts for annotating skeletons depends not only on accuracy, but also efficiency. We define efficiency as being both faster and less expensive compared to an expert.

Time

For reference, an expert can annotate 30 to 60 images per hour (depending on difficulty) with an average of about 50 images per hour. In addition, an expert is expected to get paid 200NOK to 1000NOK (25USD to 128USD[2]) per hour. This section evaluates the crowdworkers performance in both of these categories.

In our crowdsourcing campaign, a worker is paid 1.10USD per task, with each task consisting of 24 images and an expected completion within 30 or 40 minutes. A summary is given in Table 6.2 which shows how a single worker or expert is estimated to perform. The table also includes an estimation of how much time a single worker or expert would need to completely annotate all the frames in the campaign, called completion estimate.

	Worker	Expert
USD/hr	2.20	25 − 128
USD/image	0.13	0.50 − 2.56
images/hr	1 − 48	50
images/day	1 − 384	400
Completion Estimate hrs	1898 − 40	38
Completion Estimate days	1989 − 6	5

Table 6.2: Comparison between a crowdworker and an expert. *Images* are for the 1898 frames in the database and an *image* for what can be done during an hour. Day and completion estimates are based on eighth hour work days, with the results being approximate.

An expert can manage to annotate 400 images in the course of an 8 hour work day. While this number is significantly larger than a single worker average of about 192, it does not represent the actual images per day correctly. One of the key components in using crowdworkers is that they are a crowd of people ready to solve problems. Figure 6.9 shows the number of images annotated per day since the start of the campaign, with the crowdworkers greatly outperforming the expert in terms of annotated images, but not by that much. Because every image is annotated several times results in the actual annotated images per day are only a fraction of the actual count. In our case, it would be only one-fifth of the actual count, as each image is tasked an average of five times. Even then, though, they still exceeds the expert performance.

[2]Conversion rate from Google Search 2015-6-9

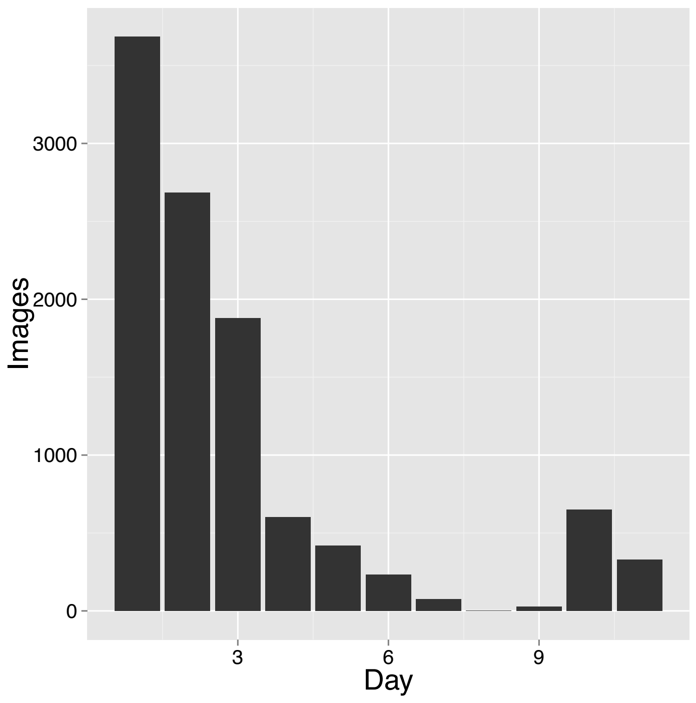


Figure 6.9: Images annotated by crowdworkers per day since start of the campaign.

Righteously, more experts can be hired, but they are far more difficult to hire, making the crowdsourcing platform both faster and easier to get the images annotated compared to experts.

Another interesting observation in the efficiency (time-consumption) in Figure 6.9 is a quick drop of images annotated after day 3. While a small decline is expected as workers either gets tired or complete their tasks, it should not be that dramatic. The reason for this is actually the query from Section 6.7 that slows down the system[3]. After optimizing this, the images per day went up to a more normal completion number.

Cost

Despite the large number of annotated images per day and low time consumption for the crowdworkers, they are actually a rather cheap workforce. With a total of 1937 solved tasks and 1267 accepted tasks in the campaign, and a total campaign cost of $1393.70USD$, means that it is costing only slightly more than the cheapest expert ($25USD*(1989/50) = 950USD$). This makes crowdsourcing an good alternative, especially considering the most expensive expert coming closer to $4900USD$ in salary. And, if not for the cost, at least for the completion time.

[3]The exact details can be found in Section 8.3.3

6.7.3 Worker Feedback

Feedback from the crowdworkers are indirectly tied to the resulting accuracy and efficiency for the annotated skeletons, as worker who enjoy or like the task are more likely to form proper work. Based on the workers who provided feedback, there is apparently a great interest in our campaign: The workers found the task to be original, interesting and even calling it a game. Some workers also participated in several of the pilots and in the final campaign, sometimes solving more than one task as well. Moreover, the workers understood the concept of using low resolution and noisy images, but they would still prefer more resolution to make annotation easier and less ambiguous. In short, the crowdworkers are more than happy to annotate skeletons, making it possible to continue use of a crowdsourcing platform for this type of tasks.

6.8 Summary

This chapter contains how annotating of skeleton joints for a annotated database are done using the `Online Training Tool` program and a crowdsourcing platform. By alleviating the workforce available in a crowdsourcing platform, a lot of work can be achieved with little cost and time. However, getting the best results from seemingly random people requires good filtering and merging of the results, referred to as quality mechanisms. The quality mechanisms in this chapter includes Gold Standard Test, a direct way to test worker effort against experts, and Majority Voting, which lets the worker themselves vote out poor-performing workers. With ours and others' experience in using crowdsourcing, and both obtaining results close to experts, makes using a crowdsourcing platform a very good approach to obtaining data that can only be set by actual people. With it, we have obtained skeletons for all the sequences designed to be used in the skeleton transfer, as part of the next chapter, Action Retrieval.

Chapter 7

Action Retrieval

The problem of performing action recognition and pose estimation in the Bagadus system are solved as a single content-based video retrieval problem. By using the data obtained from Chapter 5 and Chapter 6, a classification algorithm using a motion similarity measure is implemented to map query sequences to annotated sequences and thereby obtaining the action label and poses by reprojecting the annotated sequence's data.

Based on the approach proposed by Efros et al. [15], we model the action, or motion, *of tracked player sequences using optical flow vectors and uses these in the motion similarity measure. The measure represents the similarity on a frame-by-frame basis that determines how similar two video sequences are.*

This chapter starts with a description of the optical flow algorithm, one of many ways to obtain the motion of a video sequence. Then, the classification algorithm used to map query sequences to annotated sequences is described, including the motion similarity measure. The most similiar sequence obtained from the classification is then used to obtain the action class and a skeleton by reprojecting it from the most similar sequence to the query sequence. Reprojection of the skeletons are described towards the end of this chapter, with the results presented at the very end.

7.1 Optical Flow

In Section 1.2.4, content-based video retrieval is presented with its two parts: video extraction and a matching function (discussed later in this chapter). The extraction of video sequences consists of tracking players, as described in Chapter 5, and encoding these sequences into feature vectors. Because we are interested in classifying actions and obtaining poses, the feature vector should represent the action.

As proposed by Efros et al. [15], the video sequences are encoded as optical flow vectors. Optical flow models the motion occurring between two consecutive frames, based on pixel movement. Other approaches exists to obtain motion of a figure, for example Kalman Filter, but Efros et al. discuss how similar optical flow is to the human's perception of movement found in the retina. Efros et al. also mention that optical flow itself is not suitable to use with noisy video, such as the one in Bagadus, but that it can work by performing additional steps to normalize and account for the noise to a certain degree, and that it is sufficient to successfully compare video sequences.

7.1.1 The Algorithm

Optical flow is the chosen method to obtain descriptors from the video sequences, which is then used by the similarity measure (matching function). Its goal is to represent the movements that a soccer player carries out.

How it is Used

There exists a multitude of optical flow implementations to estimate the apparent motion, depending on what kind of motion that is to be obtained, e.g., car movement, actions or camera tilting. Nevertheless, they all work on the same basic formulation [18]: A pixel at (x, y) has an intensity I at given time t. The motion, or optical flow, has the movement $(\Delta x, \Delta y)$ for a time difference Δt, which describes the pixel movement. Calculating the movement is done by taking the starting pixel and match it against several different pixels in the subsequent image and the two pixels are said to match when the following brightness constancy constrain holds: $I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$. t can be greater than one, but real-time scenarios or consecutive video frames have it set to 1. The main differences between all the optical flow algorithms is how they estimate $(\Delta x, \Delta y)$.

To summarize, optical flow models motion as vectors by tracking pixel intensities, working under the assumption that intensity for a single pixel found in two different images is the same, on the same object, in both images, as illustrated in Figure 7.1.

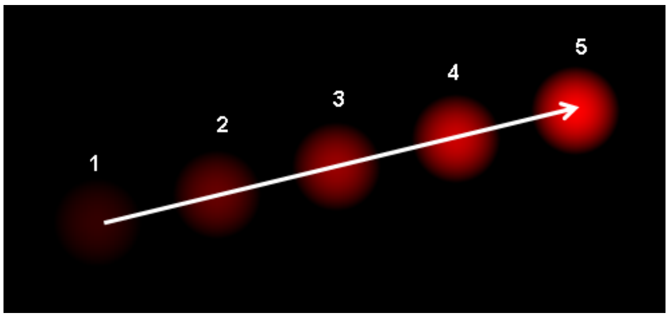


Figure 7.1: Example Optical Flow vector obtained for a moving circle across five frames. Image taken from http://upload.wikimedia.org/wikipedia/en/1/10/Optical_flow_example_v2.png.

How it is Done

Optical flow can be estimated with many different kinds of algorithms and input sources, but the choice of algorithm also depends on the type of desired output, i.e., a per-pixel or per-feature basis [28]. The first type, per-pixel, estimates the optical flow vector for every single pixel in the image and is commonly referred to as dense optical flow. On the other hand, per-feature, or sparse flow, is found by using a set of distinct features found within an image. These features can be anything from randomly selected areas (hard to estimate the movement of) to particular patches of color or edges (easy to estimate the movement of). A per-feature approach also calculates a vector for every feature, but the result set is sparse. Because it is desirable to obtain a dense optical flow, as it better models all the atomic actions, and use of the optical flow algorithms provided by the OpenCV library is necessary, it narrows the algorithm choice down to two candidates: Lucas-Kanade and TV-L_1. Experimenting with both candidates gave a clear indication that TV-L_1 matched best with the current video source, which is a discrepancy from Efros et al. [15] who uses Lucas-Kanade.

The implementation used here uses OpenCV's $Duality$ *Based* TV-L_1 optical flow function. The papers defining the algorithm can be found in the API and can be summarized quite nicely: TV-L_1 is a shorthand notation for *Total Variation Manhattan Distance*. The algorithm is based on the pixel intensity variation of an image (hence the TV) with Manhattan distance (L_1), but utilizes additional unknown variables to increase the method's noise tolerance and ambiguous pixels handling, i.e., it attempts to add regularity (consistency) into the equation. These two unknown variables that must be solved is what makes the algorithm a $duality$. Despite attempting to force regularity into the result vectors, by abstracting away some noise, this also causes it to be too restrictive. To prevent this problem, it allows for a slight error in the computation. The end result is an optical flow algorithm that is robust to illumination changes, occlusions and noise, which is ideal for the Bagadus system.

The alternative algorithm, Lucas-Kanade, while not as robust as TV-L_1, still performs well in most circumstances. Its inner workings is not that far from the original optical flow algorithm specification, but takes into account neighboring pixels instead of just one. It is capable of computing both dense and sparse optical flow vectors, where the sparse set can even be specified as a set of specific features. Using a set of specific features is quite useful when tracking objects, but not as much when attempting to estimate the whole motion occurring in a video sequence.

7.1.2 Region to Encode

How optical flow is applied has a large impact on the type of data the resulting vectors contain. The three different ways are listed below, where the primary difference being whether a player's intrinsic action or movement across the screen is most important:

1. To the original captured video frame (full frame, Figure 7.2a)
2. To the player of interest (player cut-out, Figure 7.2b)
3. To the player of interest with background removed (Figure 7.2c)

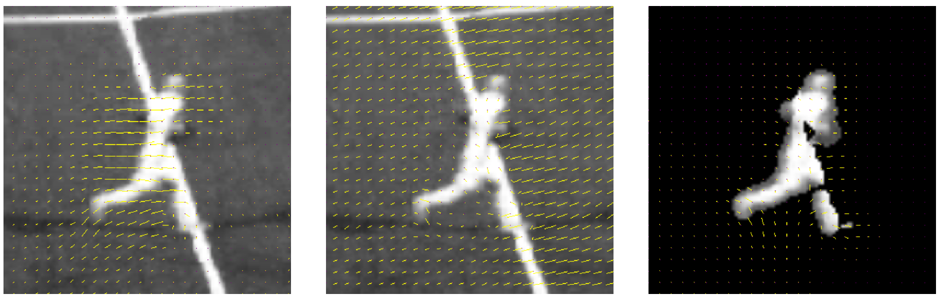

(a) Full frame (cropped to selection)   (b) Region of the frame   (c) Region of the frame with background removed

Figure 7.2: Comparison of dense flow on the three different methods.

Option 2 is not usable at all, because whenever a player crosses any of the white lines on the field, the line is included in the optical flow estimation. This occurs because the Region of Interest (ROI) is moving across the field with the player, causing the background to move across the frame. Optical flow then computes the flow for the background as well, including the white marker lines, which results in abberrant feature vectors regarding our problem and option 2 is therefore discarded.

The remaining options 1 and 3 both gives correct optical flow vectors, but with a slight difference. Option 1 gives the flow of the entire frame, but also contains the movement of the player across the field. That is, both limb movement and location difference is included in the vector.

On the other hand, option 3 only consists of the internal movement difference between two frames. It discards the additional movement caused by physical movement. One could potentially add the positional difference between previous and current frame, but it is not guaranteed to obtain the same results as option 1. However, only option 3 is used as the intrinsic movement is the desired output for the motion descriptors.

The selected region to apply the optical flow estimation also underlines the need for the player tracking algorithm presented in Chapter 5. Moreover, many other reasons are found in the action classification itself.

7.2 Action Classification

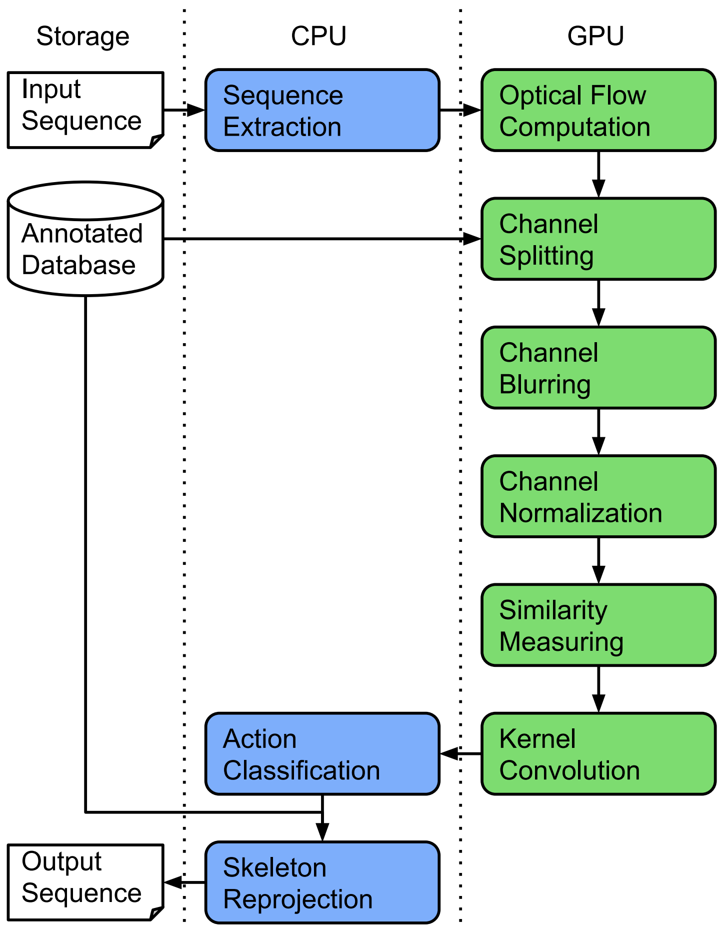


Figure 7.3: The complete pipeline for action classification and skeleton reprojection, with data in white, CPU stages in blue and GPU stages in green. The prerequisites are also included in here, which consists of the `Sequence Extraction` and `Optical Flow Computation` stages.

The second part of content-based video retrieval is a matching function that maps query sequences to annotated sequences. This matching function consists of measuring the similarity between two sequences, based on the optical flow result and computing the frame-to-frame similarity. This solution is based on the proposal by Efros et al. [15], but the authors do not include all details and because of this additional experimentation is required to complete the classification algorithm. The most similar video sequence in the annotated database is then used to define the query action class and used to reproject the skeleton.

The action classification algorithm is designed as a pipeline which is shown in Figure 7.3. The pipeline requires that the query sequence is normalized according to Chapter 5 and has its optical flow vectors computed. Secondly, an annotated database, as the one obtained in Chapter 6, containing video sequences encoded as optical flow vectors with a corresponding action class and skeletons for every frame is used to reproject skeletons into query sequences.

The rest of the pipeline starts with taking the input flow vectors and transforming them into a matrix format. It then uses the transformed matrices to obtain frame-to-frame similarity of all frame combinations and a kernel is then applied to get the temporal information encoded into the similarity. Based on a Nearest Neighbor algorithm, the highest scoring sequence is found and is used to obtain the action label and pose for the query sequence. Because both optical flow and the original video frames are used frequently in this pipeline, we differentiate between frames from video sequences and the encoded optical flow vectors for a frame in a matrix by using *frames* for the original frames and *flow fields* for optical flow vectors.

7.2.1 Channel Splitting

The first step is to split the flow fields into four separate channels. Each channel consists of the absolute magnitude of a vector at a particular pixel and holds only vectors having the same range of directions and the channel.

How it is Used

It is important that motion in a flow field is compared with motion of same direction, not with opposing direction. If the flow field is not split into the different channels, then comparison of the optical flow vectors would result in false positives during classification, because the opposing vectors would neutralize one another. It is, of course, possible to take this into consideration when computing the frame-to-frame similarity later on, but it would increase the complexity of the algorithm compared to splitting up the channels. Therefore, the flow field is split into four channels to prevent negative flow vectors from being present.

How it is Done

The flow field is modeled as a two dimensional array, whose every entry in the array is an array with two scalars (i.e., x and y). The element array defines the optical flow vector as the number of pixels required to traverse in order to get to the next position the pixel occupies (current position is origin, everything else is relative to this). The directions are then divided into 90-degree partitions, where each pair of positive and negative values of x and y denote the direction, resulting in the four channels F^{++}, F^{+-}, F^{-+} and F^{--}. Every channel has the same size as the initial flow field and channels that contains negative values are inverted to obtain positive values. Alternatively, more than four directions can be used, for instance 45-degree separation, but the 90-degree provides ample sparsity and prevents calculations with negative values (i.e., prevent scalar product between two vectors to be null). Sparsity is important in order to get distinguishable values that can be used for comparison.

7.2.2 Channel Blurring

After the optical flow vectors for a frame has been divided into four separate channels, they all need to be softened by blurring. Blurring is achieved by applying a Gaussian Filter to each of the four channels individually. This is another reason for wanting to divide the four channels: to avoid smoothing across negative and positive values, which could potentially result in magnitudes being smoothed out to the point where they become zero.

How it is Used

When the blurring is applied, it reduces the hard borders between each value, creating a larger surface of magnitude values instead of a few points. The flow field channels are then more tolerable to small misalignment in actions. Without the blurring, small misalignment in the flow fields can cause the matching to reject a pair, because the vectors did not align up perfectly. In other words, the channels becomes less absolute and will potentially match with more query sequences (i.e., positive matches). With more positive matches one gets a higher change of finding a matching sequence if the query sequence proves difficult to classify with the given trained data. Minor differences becomes tolerable, while still having the best matches getting the best score. This holds true, because a single vector that matches exactly between two channels still do so even if the channels where to be blurred with an equal amount.

How it is Done

The Gaussian Blur is described in Section 5.1.3 and is therefore not described here. Instead, we focus on the balance concerning the amount of blur to apply. In context with the Gaussian Filter, it can be translated to how large the blurring matrix should be. If the matrix is too large, then the amount of detail vanishes. Without details, the similarity certainty between any two pair of sequences becomes more questionable, and, with enough blur indistinguishable. On the other hand, if not enough blur is applied, then the effect of smoothing out values makes it less tolerable to slight differences and noise. In a nutshell, it comes to a compromise between being able to tolerate errors and offset and completely wash out distinguishable features, with discrete features being the most important for the frame-to-frame similarity comparison. By experimenting with different sizes with the default Gaussian filter provided in OpenCV, we ended up with a matrix size of 5×5.

7.2.3 Channel Normalization

The different flow field channels contains a varying range of values, i.e., the optical flow vectors have different magnitudes, depending on the pace and the action being performed. While different actions are to be grouped into separate classes, it is desired to not have the pace of the action affect the choice of the class. Because of this, channel normalization is required.

Normalization could be applied at a stage in the pipeline: over the input flow field before they are split into channels, after they have been split into the channels, or after the channels have been blurred. However, because blurring causes changes in the magnitude of the vectors, it is best to do normalization after blurring the individual flow channels.

How it is Used

There are two reasons to normalize channels: The first reason is to avoid sequences with large optical flow magnitudes from always being measured as the most similar. Because the similarity measure (later in the pipeline) computes the scalar product between two unit vectors, it will always have the sequence with the largest magnitudes for the optical flow vectors be the best matching (i.e., highest scoring). The magnitues are a result of how fast a motion of a limb is, which can vary within the same action (e.g., kicking the ball can be done with different strengths). Since actions of the same class is to be classified together, independent of the pace the action is performed, it requires normalization. The second reason is to have the similarity measure later on to be in the interval $[0, 1]$.

How it is Done

Normalization is done by calculating the mean magnitude of a channel and dividing it by this value, and it is done for every flow field for every frame in sequence. Alternatively, the mean could be computed from all of the flow field channels (i.e., F^{++}, F^{+-}, F^{-+} and F^{--}) in a frame, but because one channel typically has more movement than the others (any limb can only move in one direction at a time), it would result in key features of the other channels reduced to almost nothing. Another alternative would be to calculate the mean for a single channel across all the frames, but with most flow field magnitudes being close to zero, it would introduce a lot of noise.

7.2.4 Similarity Measuring

At this stage, the video sequences (both a query sequence and an annotated sequence) have had their optical flow descriptors normalized and are ready to be compared. The comparison is done using a similarity measure, which is implemented as a scalar product between two flow fields.

How it is Used

The feature matching function is designed to calculate a score for every single frame pair between the query and annotated sequences, resulting in a matrix where the number of rows and columns is equal to the sequences' lengths. This matrix will then, in a later stage of the pipeline, be used in a classification algorithm to determine, out of all the annotated sequences, which sequence is the most similar to the query sequence. Calculating a single point, or frame pair, in this matrix is done by using the motion similarity presented in [15].

How it is Done

The frame-to-frame similarity formula is given in Equation 7.1. It computes the scalar product of the flow fields for every frame pair (i, j), with the ranges defined in I, to obtain the frame-to-frame similarity matrix S_{ff}. It comprises the sum of the channel product between two channels a and b, corresponding to one of the flow channels (e.g., F^{++}). However, because the flow fields are modeled as matrices, a small transformation is required to have the matrices described as vectors. This is done by taking each column in a matrix and stitching them underneath one another to form a single vector. The scalar product between a and b is then expressed as Equation 7.1.

$$S_{ff}(i,j) = \sum_{c=1}^{4} \sum_{x,y \in I} a_c^i(x,y) b_c^j(x,y) \tag{7.1}$$

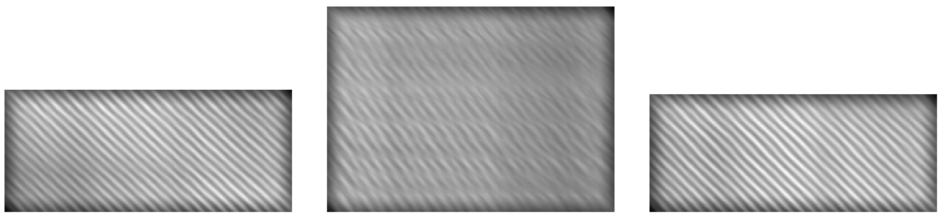
The resulting matrix S_{ff} only takes into account the temporal and spatial information of a single frame and not on the entire action. Since two identical frames can come from different actions, with the only way of distinguish them is by looking at the previous and next frames, it is required to also include neighbors' frames information to S_{ff}. This is done in Equation 7.2 and results in the matrix S having a temporal span (neighbor frames' spatial and temporal information) of T.

$$S(i,j) = \sum_{t \in T} \sum_{c=1}^{4} \sum_{x,y \in I} a_c^{i+t}(x,y) b_c^{j+t}(x,y) \tag{7.2}$$

This can be improved further, as Efros et al. [15] point out: The similarity between two sequences occur along the diagonals. This similarity is a result of how the matrix S_{ff} is constructed of (i,j) flow fields, i.e., frame pairs. Iterating though both the sequences i and j at the same time (i.e., increment i and j on every iteration), it is the same as traversing the matrix S_{ff} diagonally. Therefore, instead of accumulating the values of all the neighboring flow channels (frame pairs), only the ones in a direct diagonal is used. However, there are times when the action occurs at slightly steeper or slopier angles than to the diagonal, to which a slight offset of the diagonal should be included in the temporal span computation.

7.2.5 Kernel Convolution

Kernel convolution in computer vision is an operation used for altering the data of an image (we have already described one of these, namely the Gaussian Filter). Using this same technique, the temporal information of diagonals (i.e., the diagonal and slightly offsets from the center diagonal) can be specified as a matrix and applied to the matrix S_{ff}, resulting in the similarity matrices in Figure 7.4.


(a) Similar classes, but not same class

(b) Different classes

(c) Same classes

Figure 7.4: Rendered instances of the matrix S, using a convolution kernel with $N = 21$ and $\sigma = 0.05$. Note that dissimilar is blurred, and the best match have the clearest diagonal lines. The query sequence is represented in the horizontal direction and the annotation sequence in the vertical direction.

How it is Used

There is one large advantages to using kernel convolution compared to statically define the temporal span. The reason is that it is straight forward to change the rules that defines the extent temporal information to be added to the similarity matrix. This is particularly useful, because this makes it easier to experiment with a large number of kernels. Since there exists no definitive way of identifying the exact kernel configuration it means that a large set of kernels must be tested to find the one most suited to the Bagadus system. In more detail, the different kernels consists of changing the temporal span, i.e., number of frames to check, and how tolerable it is to offsets from the diagonal, i.e., how much difference there can be in the pace. Whichever kernel produces the most accurate action classification is the one deemed most suitable.

How it is Done

A kernel K is created by taking two parameters: N and σ. N defines the range of the temporal span to include, i.e., the number of previous or following (a Gaussian law of standard deviation σ) frames to include into the computation with σ defines the offset from the diagonal that is to be included in the kernel. The construction of the kernel then constitutes of creating a $N \times N$ matrix and then produce lines across the diagonal in this matrix. The lines have a common center point in the center of the matrix, with each line being a few degrees offset from the diagonal, with the amount specified by σ. These lines contains the weight of the matrix elements, with lines closer to the diagonal weighted higher than the ones further away. Overlapping lines' value for a given matrix element are then combined and when all the lines are created, the matrix is normalized so that the sum of all the elements equals to one. Optionally, Gaussian blur can be added. A kernel with $N = 21$ and $\sigma = 0.05$ is rendered in Figure 7.5, which includes Gaussian blur.

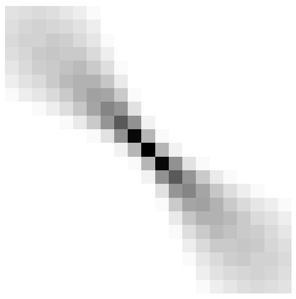


Figure 7.5: Visual Representation of kernel K with a temporal span of 21 (frames) and a diagonal variance of 0.05 (σ).

7.2.6 Action Classification

Using the matrix S, a score for a given comparison between the query sequence and an annotated sequence is obtained. Using the *Nearest Neighbor Algorithm* for classification, the annotated sequence with the highest computed score is the one that is the most similar to the query sequence.

How it is Used

The classification uses the frame-to-frame similarity matrix S to produce a score consisting of the sum of all the matching frames. In result, the optical flow vectors are directly used to define the similarity which, in itself, consists of the action performed in a frame. Consequently, if either the resulting optical flow vectors or player tracking normalization obtained do not obtain enough distinctive features, the classification will fail. For example, the action standing does not have any motion, which results in a series of empty flow fields for every frame. In this case, the classification behavior is impredictable (i.e., random). Whichever annotated sequence has the highest similarity score ends up being the sequence to which the action label is obtained from and the skeleton used in the reprojections.

How it is Done

A sequence is already compared against another sequence through the kernel K, with the comparison results (i.e., similarity measure) in matrix S. Choosing the most similar video sequence in the annotated database is then done by finding the maximum value M in S for every sequence in the database and then take the sequence with the largest M value. This sequence, found by a *max-of-max* computation, is then used to define the action for the query sequence and reproject the skeletons into the query sequence's frames.

However, there are two shortcomings in the max-of-max approach. Firstly, the comparison is only valid for the same temporal span as T and not across the entire sequence (although, the classification accuracy is more or less the same as the one we propose in the next section). Secondly, as a result of only using the maximum in S, the skeletons for reprojection at a later stage might not achive the best result: Because skeletons are reprojected i.e., transfered directly from the annotated sequence onto the query, they depend on the most similar frame to be as similar as possible to obtain the most accurate skeleton. With max-of-max, only the maximum accuracy of a *single* frame/skeleton is chosen, however, we want to maximize the number of similar frames found across *all* frames in a sequence to have as many high-accuracy skeletons as possible.

We use a similarity score that is computed from the highest matching frames for every row in the matrix S. This is done instead of summarizing the entire row, because the sum of every row can result in an overall better score than just using the highest matching frames. In other words, we are only interested in cases where the motion is indisputably similar and ignore motion that is approximately similar, but not exact. Once the score has been summarized, it is normalized by dividing the score with the length of the annotated sequence. The final classification score then ensures only the most similar sequences are classified together and that the length (number of high matching frames) does not affect the overall score. The scoring is given in Equation 7.3, with I

containing the number of rows and J containing the number of columns.

$$Score(S) = \frac{1}{I} \sum_{i \in I} \max_{j \in J} S(i, j) \tag{7.3}$$

The Nearest Neighbor algorithm can be extended to use n best matches results instead of using a single, best scoring sequence. The action labels that most of these n sequences have then becomes the final action class (but only the highest scoring among them can be used for skeleton reprojection). This could improve the classification results, but it also requires a sufficiently large dataset. In the case of n sequences, at least $n/2 + 1$ annotated sequences of every action should be present to guarantee that every action can have the majority (a proof by induction that we do not include here, as it is not used). Alternatively, weights can be applied to the voting to account for multiple suitable classes and cases with too-little data.

7.2.7 Skeleton Reprojection

From the most similar annotated sequence it is possible to extract the skeleton and transfer it into the query sequence. The skeleton is chosen from the annotated frame that has the highest optical flow similarity with the novel frame. This is found by checking each element in matrix S for highest scoring values.

How it is Used

The annotated sequence with the highest score has been obtained because it has optical flow vectors with similar directions at the approximate same place and time within a sequence. Moreover, the individual frames with the highest similarity score has the most equal motions in that particular time interval. Since these two frames are closely resembling one another it means that the skeleton, or pose, found in the annotated sequence can be used to define the pose for the query sequence. In essence, the annotated sequences' skeletons are *reprojected* onto the most similar frames found in the query sequence.

How it is Done

The skeleton reprojection works by taking the skeleton annotations, which consists of pixel coordinates according to the tracked ROI, and place these joint locations in the query sequence. To correctly align the skeleton onto the query sequence, the annotated and query frames are placed on top of each other so that the center of each ROI is in the same location. After that, the joint positions can be directly transferred to the query frame. We also adjust the height of the reprojected skeleton, by scaling it uniformly around the center. Specifically, the scaling factor is found by computing how much the annotated frame's ROI must be up-scaled or down-scaled to match the query ROI.

Even though a pose has been obtained for the query sequence, it is a best-match and not a perfect match search, meaning that there will be discrepancies between the frame and the pose. Additional adjustments of the transferred skeleton is required to obtain a precise pose, but is left as a later improvement, because it requires additional segmentation of the video frames to correctly adjust the skeleton. Although, the results could be improved somewhat by moving every joint outside the foreground mask the minimum amount of pixels required to be inside the mask. This should improve situations where the skeleton is slightly different in size (caused by difference in field positions when captured), but is not done due to the inconsistency issues from the background subtraction. Overall, the final skeleton sequence can be exported to any data format required.

7.3 Results

There are two sets of results obtained from the classification: action labels and skeletons. Before discussing any of the results, a couple of measurement methods is presented in order to properly define the classification results.

7.3.1 Classification Terms

When operating with classification, there are five terms that needs to be understood: false positives, true positives, true negatives, false negatives and confusion matrix. Each of these terms describes how correct a particular classification is compared to the correct solution.

Accuracy Measurements

The first four terms are used for computing the accuracy, or correctness, of a classifier. The first, *true positive*, is an instance that is successfully placed in its belonging class. The main goal of every classifier is to have all the input queries classified entirely as such. However, there are times when *false positives* occur in the result. These are instances where the input query is rejected from a class, despite the fact it should be included. Not to be confused with *false negatives*, which is an included sequence even though it should not. The last one, *true negatives*, are instances that are not to be included in the class and successfully rejected as such. With all of these terms, it is possible to discuss upon a classifier to describe the quality of the algorithm and in which aspects it succeeds or fails. The most common quality measure is accuracy, computed from the formula $(true\_positives + true\_negatives)/total\_elements$. The higher this score is close to one, the better. There is a second type of statistics, called recall, but it offers no usable information to this thesis. The reason being that it measures the amount of sequences found in the dataset compared to the actual available, but considering we run through all the sequences, it makes this measure useless. The same goes for $true\_negatives$, which does not provide any useful statistics than what $true\_positives$ already provides.

Confusion Matrix

True positives, false positives, true negatives and false negatives are best used with a two-class classification problem. When more that two classes are considered, a *confusion matrix* is a better way to both visualize and evaluate classification results. A confusion matrix is a two-dimensional table with expected classes along one axis and actual classification results on the other. For a perfect classification algorithm, all the classification results should be along the diagonal from top left to bottom right (i.e., every classification result is a true positive). The benefit of using this matrix over the regular accuracy measurements above, is that in addition to showing $true\_positives$ it shows $false\_positives$ in an equal degree. Identifying the instances of $false\_positives$ can aid in determining the factors involved in a failing classifier, as we can see in the next section.

7.3.2 Classification

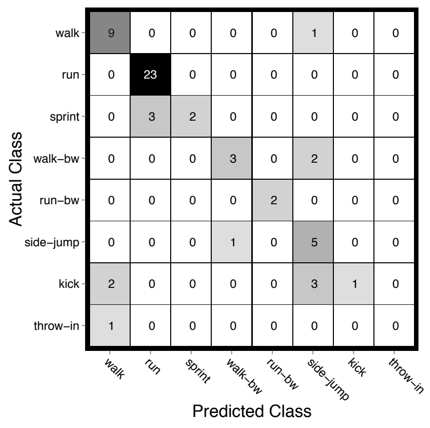
To determine the accuracy of the classifier presented in this chapter, we extend the set of sequences used in Section 6.7 with an additional 42 sequences. Also, because Efros et al. [15] do not specify clearly the kernel configuration, and to better adjust the similarity measure to our scenario, we run the classifier with a combination of kernel sizes sigma values. Table 7.1 displays an approximately accuracy measure, calculated from $true\_positives/total\_elements$, obtained from the different kernel configurations.

$N\backslash\sigma$	0.0	0.2	0.5
9	76%	70%	70%
13	76%	72%	72%
15	74%	72%	70%
21	78%	72%	70%

Table 7.1: Summarized classification accuracy for different kernel sizes and sigma values.

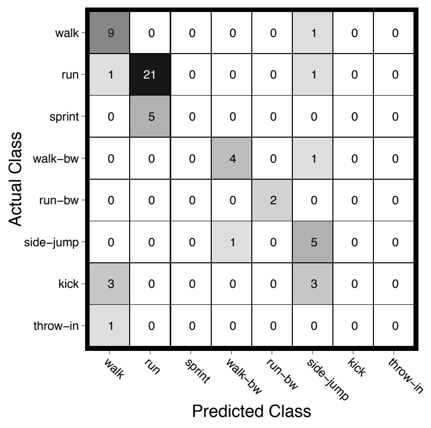
With a relative large variance in kernel sizes and sigma values, a larger variance in the table was expected. The small variance may result from using the same player for all sequences in the dataset, which also leads to consistency, i.e., the speed an action is performed at remains consistent between different sequences of the same class. Alternatively, the consistency can come from the selected test-set, as it contains only those sequences which can be certain to produce proper tracking results (and thereby also nice optical flow vectors).

In other words, the sequences found are already matching really well to one another, independent of the kernel used.



Actual Class \ Predicted Class	walk	run	sprint	walk–bw	run–bw	side–jump	kick	throw–in
walk	9	0	0	0	0	1	0	0
run	0	23	0	0	0	0	0	0
sprint	0	3	2	0	0	0	0	0
walk–bw	0	0	0	3	0	2	0	0
run–bw	0	0	0	0	2	0	0	0
side–jump	0	0	0	1	0	5	0	0
kick	2	0	0	0	0	3	1	0
throw–in	1	0	0	0	0	0	0	0

(a) Confusion Matrix for $N = 21$ and $\sigma = 0.00$



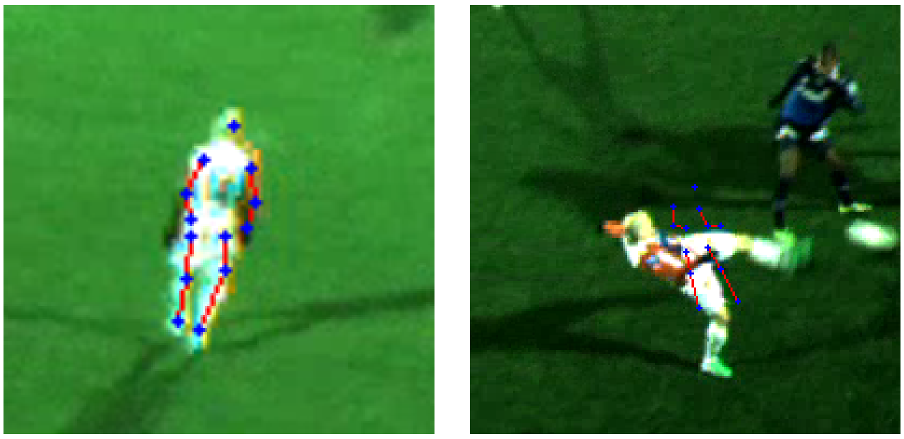
Actual Class \ Predicted Class	walk	run	sprint	walk–bw	run–bw	side–jump	kick	throw–in
walk	9	0	0	0	0	1	0	0
run	1	21	0	0	0	1	0	0
sprint	0	5	0	0	0	0	0	0
walk–bw	0	0	0	4	0	1	0	0
run–bw	0	0	0	0	2	0	0	0
side–jump	0	0	0	1	0	5	0	0
kick	3	0	0	0	0	3	0	0
throw–in	1	0	0	0	0	0	0	0

(b) Confusion Matrix for $N = 21$ and $\sigma = 0.05$

Figure 7.6: Confusion matrices for a selection of kernels, with the number defining the number of instances for that classification pair.

Another factor that limits the accuracy for every kernel configuration could be the expert labeling of the sequences. This is especially true for the "run" and "sprint" actions, which are only distinguished by the leg strides and visual speed across the frame rather than an actual metric, e.g., calculated velocity of a player for consecutive frames using ZXY positional data. Small subjective differences can cause accuracy loss due to miss-labeling, but considering that most of the time these actions are correctly classified makes it sounds more plausible in an error from the classification algorithm. In particular, the channel normalization stage in the pipeline can cause miss-classifications. While it is designed to allow mapping of similar actions to the same class independent of pace, it might negatively affect the distinction between "run" and "sprint", as magnitude information is lost during the normalization.

Looking into one of the best and one of the worst confusion matrices in Figure 7.6b and Figure 7.6a respectively, a few more observations can be made. First of all is the class distribution of sequences: with running being predominately the most common action in our dataset and is also the most common in most, if not every, soccer game. This causes other, non-running actions to be classified as false positives, i.e., as running. An example for this is "side-jump", which is commonly mis-classified as "run", or, the inverse: "run" being classified as "side-jump", e.g., Figure 7.7a. A possible explanation for this is how optical flow is calculated: It does not take the direction the player is facing into consideration and with both side-jumping and running performing a lot of scissor-like leg movements, while remaining relatively still with the remainder of the body, it can end up having very similar optical flow vectors. The observed camera angle may also have an effect here, making the two actions appear similar with a given perspective. Similarly, "side-jump" is often put in cases where "kick" would be the proper class, caused by the same leg-movement spread e.g., Figure 7.7b, which has incorrect class and thereby incorrect reprojection of the skeleton and even fails to scale the skeleton correctly, although the latter is caused by player tracking failure due to background subtraction not being correct.


(a) "Walk" classified and with skeleton reprojected using a "side-Jump" class.

(b) "Kick" classified and with skeleton reprojected using a "side-jump" class.

Figure 7.7: Results from classification and skeleton reprojections. Blue points marking joints and red lines marking connecting limbs.

On the other side, though, the "walk backwards" action is rarely confused with its regular "walk" (forwards) class, despite having very similar vectors when facing direction is ignored. Still, human legs do not move in the same manner both ways and this might be just enough for it to be distinguishable by the similarity measure. Also, an accuracy of 78% might not be a bad percentage if the final reprojected skeletons are accurate. However, even if the skeleton reprojection is spot on, there might be an issue for programs using the class label for statistics. In short, if the action classification without the skeleton transfer is desired to be used in automatic annotation of the players during a match, a much higher accuracy is desired.

While still a lot more research and development is necessary to firmly define the causes for all the miss-classifications, and solutions for correctly classifying the remainder 22% of the video sequences, it is far from a random guessing of the correct class, which shows that the chosen action classification algorithm works to a certain degree.

7.3.3 Reprojection

Since the best accuracy is achieved with a kernel of 21×21 and $\sigma = 0.0$, it is used as the only kernel when obtaining the skeleton reprojections results. By comparing the reprojected skeletons obtained with the classification and reprojection pipeline towards to the same sequence's annotated skeleton, the euclidian distance between the joints can be computed, measured in pixels, and the *reprojection error* can be plotted. Figure 7.8 presents this plotting, by first finding the mean of every frame in a sequence and then display the median, maximum and minimum value for all the frames. Additionally, the graph ignores any obscured points. The graph also includes the classification fail/pass results for comparison. Because only sequences annotated by the crowdworkers are used, the number of sequences are limited to only 27.

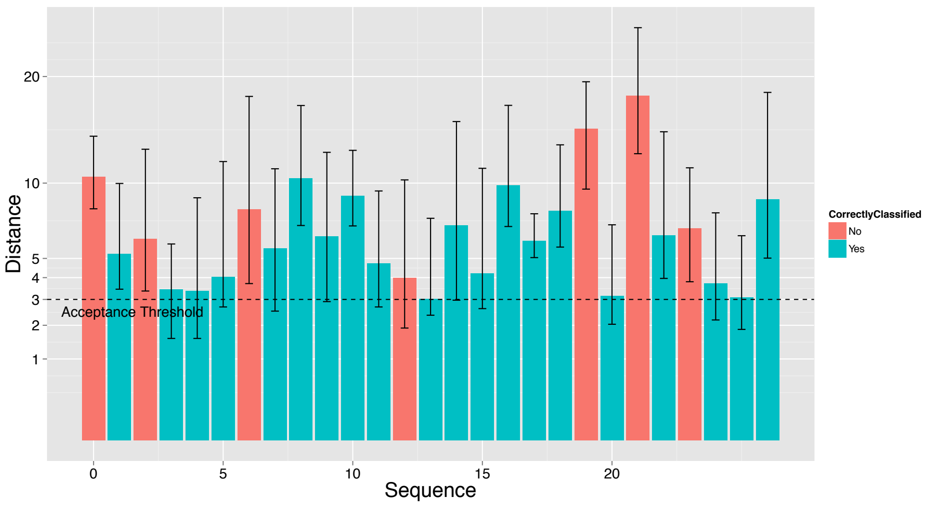


Figure 7.8: Skeleton reprojections compared to ground truth, with the difference measured in pixels. The graph shows median values with maximum and minimum bars. Everything below the acceptance threshold is considered acceptable, but is only valid for these sequences.

In order to have any meaningful way to speak of accurate or inaccurate reprojection results, we define an accuracy threshold of 3 pixels. 3 pixels deviance is within visually correct placement of joints of the 27 sequences used (cf. Figure 6.6 on page 69), but other sequences may or may not have a smaller or larger deviance threshold.

With a accuracy threshold in mind, one quickly observes that the reprojection results are not as precise as we would like them to be, but they are not that far off either. Looking at Figure 7.9 of best and worst-case representations, a few observations can be made. Firstly, almost every joint location of the correctly classified sequence has at least one frame in which the joint is no more that one pixel from its correct placement. Moreover, the mean repro-

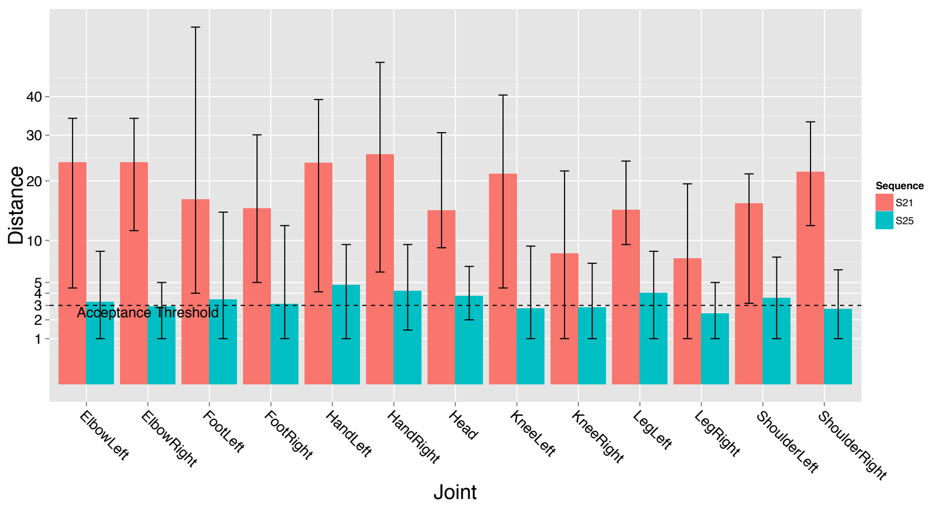


Figure 7.9: Joint reprojection error of Sequences S21 and S25, with the difference measured in pixels. The graph shows median values with maximum and minimum bars. Everything below the acceptance threshold is considered acceptable, but is only valid for these sequences.

jection error remains quite consistent for all joints. However, what eventually makes the overall precision of the skeleton transfer being low is caused by large fluctuations in distance of joints during a sequence. For example, Figure 7.10 shows both close and far-off (greater than 3 pixels) reprojections compared to the correct placements (the single dip is caused by the joint being obscured).

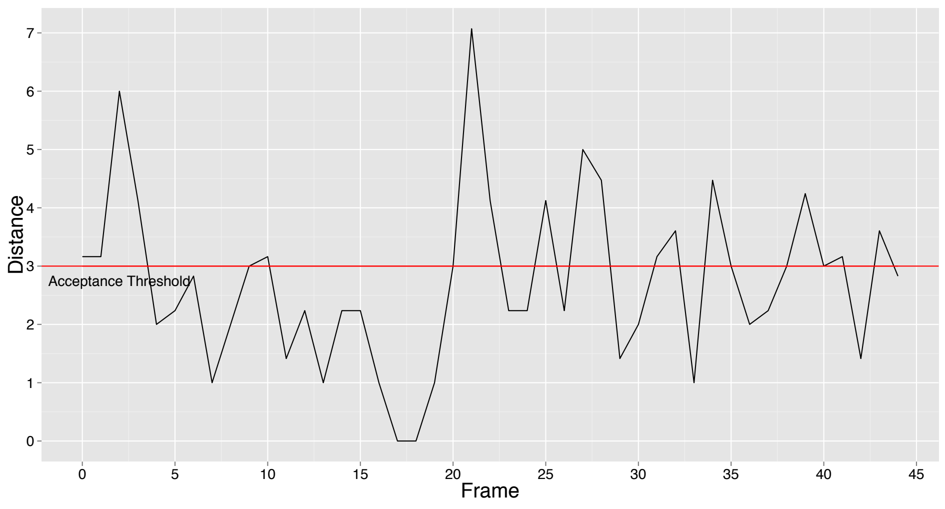
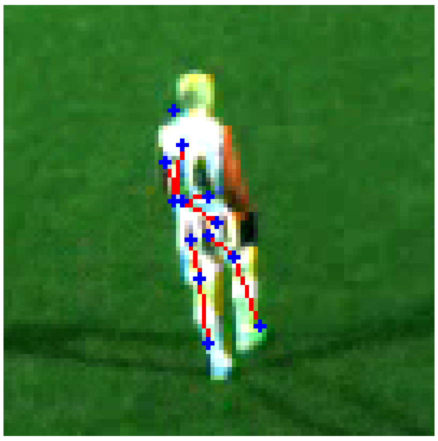


Figure 7.10: Joint Reprojection error of Sequences 25's Right Knee. The red line is an acceptance threshold.

A couple of factors affects the results and causes the accuracy of the reprojected skeleton to be lower, namely the classifier itself and the fact that no adjustments are made. With the classifier, despite that it finds the correct action most of the time, does not take orientation of the players and distance from camera into account. Orientation is important, because the optical flow can appear very similar, but, for example, a 45-degree difference in running direction results in misalignment on the torso and arm, but not on the legs as shown in Figure 7.11a. This could potentially be resolved by adjusting the skeleton after reprojection, as currently the skeleton is only copied from the annotated sequence with only size-adjustments (which at times fails to correctly resize the reprojection to the new image). Modifying the skeleton after initial transfer is done in several other algorithms, e.g., Zhang et al. [59] uses a two-step design with the second step being a refinement process, and should definitively improve, or at least reduce, the small misalignment. Increasing the number of sequences in the annotated database with a greater variety should in theory provide sequences with closer similarity than the ones currently used.

Even without a large database of annotated sequences, orientation-conscious classification and post-adjustments of skeletons, the reprojections can at time be quite accurate. Figure 7.11b shows a best-case scenario of skeleton transfer and is a perfect exsample for showing the viability of using content-based video retrieval and a classification algorithm for obtaining skeletons, with the initial results being quite promising. However, without an attempted 3D re-construction, deciding if the skeletons obtained can be used in an eventual, e.g., freeview application, makes it difficult to determine for sure how good the skeletons are, but that is outside the scope for this thesis.


(a) Skeleton reprojected with a 45-degree facing direction error.

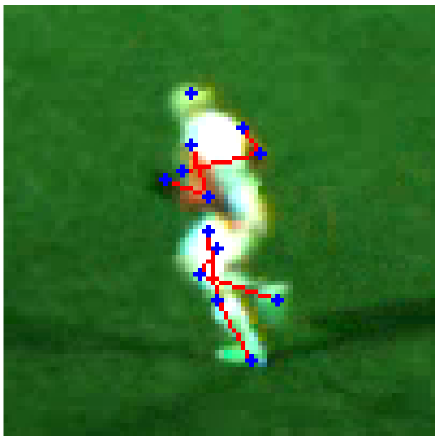

(b) A pixel-perfect skeleton reprojection.

Figure 7.11: Examples of skeleton reprojections, with blue points marking joints and red lines marking connecting limbs.

7.4 Summary

In this chapter, we have described the optical flow algorithm used to obtain the motion descriptors for every frame. Using these descriptors and the algorithm proposed by Efros et al. [15], a motion similarity function has been implemented that matches input sequences with a database of actions. Matching sequences is then used to transfer, or reproject, the skeleton and class label onto the query sequence. The skeleton is also exported and usable by other applications, for example in use in digital scene reconstruction or analytic engines. With this approach, 78% of all sequences were correctly classified and up to pixel-perfect poses were estimated (i.e., reprojected). Although inital results are promising, more research is required to determine if this process is suitable for use in the Bagadus system.

Chapter 8

Implementations and Optimizations

Implementing the programs described in Chapter 4 is done using C++, the primary language found in the Bagadus system. Yet, developing everything from scratch is both time-consuming and error-prone, therefore a set of C++ libraries that contains the required functionality are used that greatly reduces both complexity and development time. The set of libraries used in this thesis extends from simple pre-processor commands for multithreading, to serialization and image processing. Additional libraries are also used in a lesser degree for the `Online Training Tool`*, which consists of javascript and PHP code.*

Even with an extensive set of libraries that exists for C++, there is still a lot of development needed for the individual programs. Most if this code is close to or equal to the described pipelines found in their respective chapters, with only the most notable implementation details described here. We emphasizes important aspect of the implementation and explore execution speed and optimizations. In summary, this chapter includes a listing of the external libraries included to support the final pipeline and programs, as well as covering some parts of implementation and optimization problems, organized according to the respective chapters. All the source code can be found at [24].

8.1 External Libraries

Quite a few of the algorithms and techniques described in this thesis can be time consuming to implement from scratch. Using libraries that have the desired functions already present is a reliable approach to implementing these functions and some of these libraries are also optimized, saving time in both code writing and execution speed. This chapter covers all of the libraries used in this thesis for the development of the programs listed in Chapter 4.

8.1.1 OpenCV

Open Source Computer Vision Library (OpenCV) [38] is an open source BSD licensed library initially developed by Intel as a research project, but is now self-supporting. The library aims at providing both common and state-of-the-art functions for machine learning and image processing, with options for acceleration on graphics processors and multicore systems, with a varying degree of optimizations. OpenCV is available in several languages, including Python, MATLAB and C++, which makes it suitable to any development environment. Additionally, the online API provides references to the implemented algorithm, which is extremely helpful for research projects.

OpenCV comes with many other benefits that are useful when working with video and image systems. With its large set of predefined classes for math and image representations, it greatly reduces the amount of new classes that needs to be written. More importantly, these classes are useful in scenarios not tied directly to OpenCV usage. A prime example is the `cv::Mat` class, which contains all required information of a single image, that can later be used on the GPU, drawn upon, displayed or computed with any of OpenCV's functions or ones written separately. The `cv::Mat` class can also be used for general matrices, and makes it very easy to display the values with color-coding, something that can be a tedious task in regular C++. As a side-note, all of the figures present in this thesis is created using OpenCV and its rendering functionality.

8.1.2 OpenMP

Open Multi-Processing (OpenMP) [37] is a library that allows for parallel programming using compiler directives (known as preprocessor directives in C/C++, e.g., `#define` and `#include`). Owned by OpenMP ARB, it is designed to be powerful yet a simple way to make code run in parallel and can be included in everything from simple embedded systems to multicore architectures. The library, or compiler directives, can be both run-time and compile-time configured and can also automatically scale the number of cores to use if left unspecified. *gcc* (and *g++*) supports its use by adding `#pragma` directly to related code and linking with the OpenMP library, making coding fast and reliable.

Code Snippet 8.1: Code for multiplying individual elements in two arrays together running in parallel using OpenMP

```
void channelProduct(float *result, float *a, float *b, size_t length)
{
    #pragma omp parallel for
    for (size_t i = 0; i < length; i++)
        result[i] = a[i] * b[i];
}
```

The most notable application for OpenMP is to parallelize `for` loops in existing code, as Code Listing 8.1 shows. It can be configured to a specific machine and its use-case, but most of the time it is sufficient to parallelize the workload of the outermost loop across all available cores. OpenMP automatically handles thread creation, merging and work division (using `pthreads` for POSIX systems), a task usually prone to errors when implemented manually.

8.1.3 Boost

Boost [7] is a collection of libraries created to extend and enhance the already existing C++ standard libraries. The collections include everything from containers not present in the C++ STL to utility math functions and serialization. Each library is created as a module that can be included as needed and appears non-intrusive in the code. Unlike many other libraries, Boost's collections are included more and more in the actual C++ standard, making it a de-facto choice for any applications requiring any of its featured libraries. The two library packages used in this thesis are *serialization* and *iostreams*.

With a large enough dataset it is not possible to contain it in system memory and usage of storage devices becomes a necessity. The same applies to this thesis, where the length of all the annotated sequences exceeds the available RAM capacity. The Boost library *serialization* is utilized to precompute the required data once and store it in a file that can be read on demand, reducing both running time and RAM usage. Boost also supports OpenCV classes natively, which allows for using `cv::Mat` class directly in the serialization.

The *iostreams* is used to get access to zlib compression, a loss-less compression than reduces the filesize of the optical flow serialization to approximately one-half or as much as one-third of the original filesize. Compression reduces file-size, but also reduces I/O time. Smaller file-sizes are inherently faster to read and decompression on modern CPU's take little to no time at all, making use of compression an easy choice.

8.1.4 FFmpeg

FFmpeg is an open-source cross platform library for recording/capturing, streaming and encode/decode video [17]. It is also offered as a stand-alone program, but we use the library and include it directly into source code. FFmpeg support most, if not all, video and audio codecs available and manages to do this by combining other libraries (e.g libavcodec for h.264) with its own contribution to obtain the final FFmpeg library. As a result of the many libraries used, it provides several approaches to the same solution depending on need.

FFmpeg is not as good as it could be though, with inconsistent API calls and deprecated functions that must still be called. Moreover, its functionality has a lot to be desired, especially when dealing with cross-platform video exports: Attempting to encode h.264 AVI in FFmpeg resulted in improperly generated container headers and is not possible to view without FFmpeg. It could also improve handling of double frames and empty frames, which at times is desired to be left in place. Despite all this, it performs well, utilizing all available resources on the computer to achieve fast encode and decode speeds. Its extensive API allows it to be integrated into any development environment and with implementations for reading and writing video already present in the Bagadus system makes it an ideal choice.

8.1.5 MeekroDB

MeekroDB [32] is a library for PHP (the language used for server-side development in this thesis) that aims at removing the unnecessary code while maintaining the same security level as if it were to be manually implemented, with additional security against all types of injection attacks. MeekroDB's syntax has a small learning curve before it can be used properly, but that additional effort pays off. First off, it automatically handles the connection to the database, which at times can be difficult with the many different execution paths and query order, removing a lot of housekeeping source code. Secondly, large queries with a multitude of input values remains both manageable and readable at later times, with each query having an array association between SQL columns and PHP variables (see Code Snippet 8.2).

Code Snippet 8.2: Registering of a new user in `Online Training Tool` using MeekroDB.

```
DB::insertIgnore('crowdworker',
                 array('sessionId'  => $_SESSION['session_id'],
                       'username'   => $_SESSION['username'],
                       'startTime'  => DB::sqleval('CURRENT_TIMESTAMP'),
                       'goldErrors' => DB::sqleval('NULL')));
```

Database interactions is extremely common with any web service, but can at times become a tedious process, especially when including typechecking of input and prevent harmful values from entering the system (i.e., strings containing SQL commands). PHP provide *prepared* SQL statements, but requires a lot of work: It starts with a regular SQL query string which it compiles to a binary format, then it binds all the input/output variables and finally executes the actual result. We use MeekroDB to aid in the development of the server-side for the `Online Training Tool`. It is also particularly useful when registering the click-points into the database, as it makes the code maintainable and easy to read compared to prepared statements.

8.1.6 Whammy

Whammy is a client-side video processing library that takes a set of images and exports them to a HTML5 video element. Unfortunately, the library has only proper support for Google's web browser Chrome, making it locked into only one web browser vendor.

The Whammy library is used when providing an action preview video in the `Online Training Tool`, consisting of images exported from the `Data Generator` program and removes the need for a tremendous amount of pre-encoded videos. This video is then used by the annotator to help judge the position of joints and to determine the action class.

8.2 Player Tracking

There is not a large difference between the algorithm described in Chapter 5 and the actual implementation, except in two occasions. First, the Gaussian Blur is applied to the input video frame before computing the foreground mask. This is done to prevent blurring of the actual foreground mask (which should only consists of foreground/not-foreground pixels and not halfway in between foreground/background), but the end results are the same. Secondly, to improve the overall performance of the classifier, a serialized file (using Boost) with the optical flow vectors is created at the same time the images are extracted (players tracked). Not only does this reduce the overall compute requirement of the classifier, by not only having to re-compute the optical flow on every run, but also the time it takes to read in the actual video. With a pre-defined serialized file with contents corresponding directly to the tracked video sequences, it is possible to read the serialized file with the indexes provided by the SQL export file, without ever having to read in original video frames.

8.3 Video Annotation

The *Video Annotation* chapter (Chapter 6) contains the details of how the skeleton data is obtained. However, because the `Online Training Tool` (and subsequent programs) uses a different coordinate system than absolute positions, it means that a conversion is required. Secondly, with a large amount of workers and tables exceeding 10000 entries, it is a necessity to have optimized queries to prevent the system from slowing down under the load. Fortunately, only one query was found to be under-performing and is the one covered here.

8.3.1 Coordinate System

The coordinate system used in the CSV files containing either the individual crowdworker's click-points or the merged result uses a relative coordinate system. It is relative because it depends on the selections made during annotation of a frame, with determining factors being the first upscale, the bounding box selection and its secondary up-scaling. The first upscale is a constant one of 2.5 times the original size. Then a bounding box is selected and an up-scaling based on this is performed. Because every worker selects a slightly different bounding box, it involves recomputing the joint positions for every new annotation of a frame found in a CSV file. The makup of the relative coordinate system is illustrated in Figure 8.1.

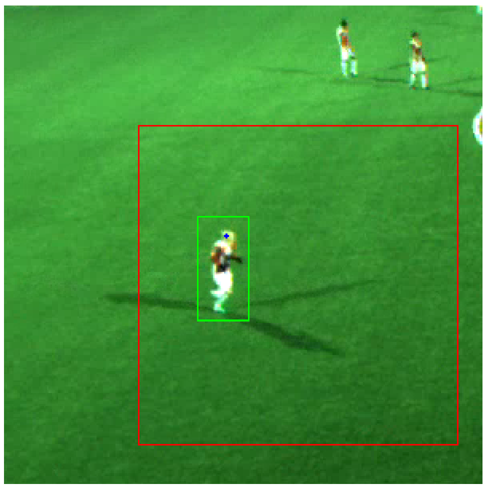


Figure 8.1: The factors determining the position of a click-point, which is relative to each individual annotation. The makeup consists of the tracker data (red), user selection (green) and click-point (blue). Click-points are relative to user selection, which is relative to the tracker data.

All the programs presented in Section 4.1.2 operates on these files by translating the relative coordinates to absolute coordinates internally. External programs can use the program `Coordinate Translator` to obtain absolute positions, which is obtained performing the steps in `Online Training Tool` in reverse. The steps the `Online Training Tool` performs to annotate a skeleton, which is also how the coordinate system is defined throughout the programs in this thesis and consists of the following steps:

1. Upscale the square input image of `size` pixels to 500 pixels to fill the canvas
2. Select a bounding box in this image, with upper-left position stored in `bb_start` and bottom-right in `bb_end`
3. Upscale the selected bounding box region to fill half of the canvas size of 500 pixels, ensuring no region is scaled outside the canvas and up-scaling factor never is below 1
4. Annotate joint positions (click-points) in the image up-scaled twice and store them in `pos`

The reverse process, to translate the relative coordinates to absolute, consist of performing the same steps but having the up-scaling factors define down-scaling instead. Code Snippet 8.3 shows a function that takes the variables in the steps above and translates `pos` into absolute pixel values, with the upper-left corner being origo.

Code Snippet 8.3: Converting relative joint positions to absolute joint positions using OpenCV

```
Point2f relativeToAbsolute(int size, Point2f bb_start,
                           Point2f bb_end, Point2f pos)
{
    // Calculate absolute bounding—box positions
    float initial_scale = 500 / size;
    float bb_x1 = bb_start.x / initial_scale;
    float bb_y1 = bb_start.x / initial_scale;
    float bb_x2 = bb_end.x / initial_scale;
    float bb_y2 = bb_end.x / initial_scale;

    // Calculate second scaling
    float bsx = 500 / (bb_x2 - bb_x1);
    float bsy = 500 / (bb_x2 - bb_x1);
    float zoom_scale = fmaxf((fminf(bsx, bsy) / 2) , 1);

    // Calculate absolute positions
    float px = pos.x / (zoom_scale * initial_scale) + bb_x1;
    float py = pos.y / (zoom_scale * initial_scale) + bb_y1;
    return Point2f(px, py);
}
```

8.3.2 Database Layout

Figure 8.2 describes the database layout used for the `Online Training Tool`. The most important tables are *trainingdata*, which contains the images to be trained and worker annotated images, and *crowdworker*, which keeps track of worker sessions and links them to the annotated images. *goldLogs* has the same definition as *trainingdata*, but is used to log Gold Sample Tests only (See Section 6.4.3). The last view, *sequenceframes*, is used in the optimized version of the sequence load-balancing query. The layout is included here for reference, as it makes it easier to describe the performance problem caused by one of the queries presented in the next section.

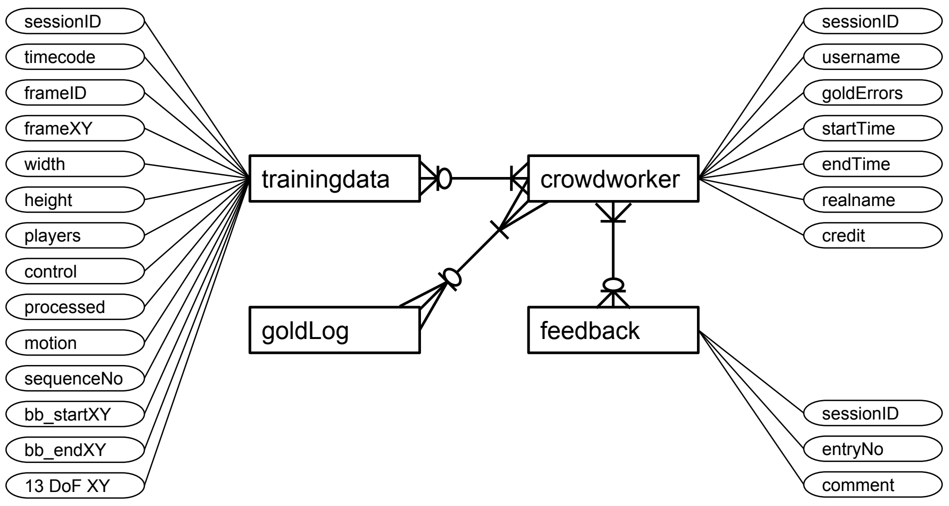


Figure 8.2: ER model of tables used in the `Online Training Tool`, using *XY* suffix where two columns are present in the database for x and y coordinates and *13 DoF* a shorthand for all the individual joint positions.

8.3.3 Query Optimization

The query found to be far too slow is the one used in Section 6.6.1 to perform load-balancing on video sequences. Originally, it was designed to operate as a single query without the need for additional tables or views, as shown in Code Snippet 8.4. However, when the *trainingdata* table started to reach 10000 entries, the query failed to produce a result within reasonable time. Because of this, the optimized query in Listing 8.5 was constructed, along with the view definition in Listing 8.6, as views can greatly increase the performance of queries [48]. Both queries obtain the same result, a number defining the number of times a sequence has been completely annotated, but the optimized version have no problems with large tables.

Code Snippet 8.4: Original query for obtaining sequence load numbers

```sql
SELECT sequenceNo
FROM
    (SELECT processed.sequenceNo, (processed.frames / lengths.frames) AS
        trained
     FROM
        (SELECT DISTINCT(sequenceNo), COUNT(sequenceNo) AS frames
         FROM trainingdata WHERE sessionID = 0 GROUP BY sequenceNo)
     AS lengths,
        (SELECT DISTINCT(sequenceNo),
            (SELECT COUNT(frameID)
             FROM trainingdata t2 WHERE t2.sequenceNo = t1.sequenceNo AND
                 sessionID != 0)
             AS frames
             FROM trainingdata t1)
         AS processed
         WHERE lengths.sequenceNo = processed.sequenceNo)
AS workload
ORDER BY trained ASC, RAND()
LIMIT 1;
```

The main issue with Code Snippet 8.4 is its frequent use of `GROUP BY` statements and it is also used in sub-queries, both of which are factors in slowing the system down. Additionally, the way it counts the frames in inefficient: it finds sequences and counts frames, rather that simply counting frames and processed times and grouping them to sequences afterwards, as done in Code Snippet 8.5. Not only is the optimized version easier to read and maintain, but it does not suffer from repeated `GROUP BY` statements, as the view is created no more than once per use of the optimized query (or less if caching is involved [48]).

Code Snippet 8.5: Optimized query for obtaining sequence load numbers

```sql
SELECT sequenceNo, SUM(trainedframes)/COUNT(sequenceNo) AS workload
FROM sequenceframes
GROUP BY sequenceNo
ORDER BY workload, RAND();
```

Code Snippet 8.6: View utilized in the optimized query for obtaining sequence load numbers

```sql
CREATE VIEW sequenceframes AS
    SELECT sequenceNo, frameID, COUNT(frameID) - 1 AS trainedframes
    FROM trainingdata WHERE sequenceNo IS NOT NULL
    GROUP BY sequenceNo, frameID;
```

8.4 Action Retrieval

The *Action Classification* program have two different ways of operations, one being direct in-memory database and the other using the serialized file created by the `Data Generator` program. While both options are still present in code, utilizing the serialized file is preferable, both due to memory consumption and running time. In addition, to further reduce the run-time of the classifier, the algorithm computing the similarity between two sequences was ported to the GPU.

This section covers briefly the consumption requirement of the in-memory database (which also underlines the decision for creating a pre-computed optical flow serialized file) and the GPU performance improvement.

8.4.1 In-Memory Database

The first implementation for the *Action Classifier* read in all video sequences and stored them in main memory. It would store the full video frame, background mask and optical flow for every frame in a video sequence. With a 1280×960 frame size using OpenCV's *CV_8UC3* format, i.e., 24-bits per pixel, it consumes $\sim 3.69MB$ of memory. The background mask is smaller, requiring only one byte per pixel[1], resulting in $1.22MB$. Finally, the flow vector is constructed out of two 32-bit floating point numbers per pixel, containing values for the x and y axis. It uses a total of $9.8MB$ of memory per frame. Adding these numbers up results in a minimum memory usage of $\sim 14.71MB$ per frame. Considering a default system memory capacity of a computer, as of writing this is $8GB$ (although only about $6-7GB$ can be used due to OS and other programs), a maximum of approximately 400 frames or 13 seconds of video can be stored in memory at any time. Without a doubt, it is not possible to store every sequence with all its data in memory.

Because of the memory limitation, the optical flow for each of the files gets precomputed during image creation in `Data Generator`. When the `Motion Classifier` program runs, it reads a video sequence from serialized data, compute over it and removes it from memory before reading the next sequence. This technique gave a large reduction in I/O operations and a slight reduction in time consumption. The compute time can be reduced even further by implementing the action classification algorithm on the GPU.

[1]This is how OpenCV implementation works. A more optimal solution would use a bitmask instead.

8.4.2 GPU Optimization

GPU optimization consists of utilizing the massive data-parallel computational capabilities of a Graphics Processing Unit (GPU) (graphics card). Because a GPU is designed to calculate and process millions of individual pixels per second, it is ideal when processing the frame-to-frame similarity measure used in this thesis. Also, GPUs support massive parallel computing with particularly good speedups for machine learning applications with vector and matrix computations [9], making it suitable for running the action similarity measure. A port of the classification algorithm was made to CUDA, nVidias GPU's computing platform and Application Programming Interface (API).

Other than porting the code to CUDA, there are no additional optimizations on CPU or GPU implementations, apart from improving the channel product computation described in Section 7.2.4 and running code in parallel using OpenMP on the CPU. Also, instead of turning the matrices into vectors with stitching and by transposing one of the stitched vectors, direct matrix as a vector computation can be made, which speeds up the computations and simplifies kernel convolution. More precisely, both matrices are computed as if they were vectors and has the same result, but with using a matrix multiplication method as shown in Code Snippet 8.7.

Code Snippet 8.7: Channel product with two OpenCV Matrices as vector product

```
float channelProduct(Mat &trained, Mat &sequence)
{
    float product = 0.0;
    for (int y = 0; y < trained.rows; y++)
        for (int x = 0; x < trained.cols; x++)
            product += trained.at<float>(y, x)
                       * sequence.at<float>(y, x);
    return product;
}
```

Additional optimizations can be added later on, e.g., SSE or AVX instructions on the CPU and ensuring memory coalescing and asynchronous memory transfers on the GPU, but this is outside the scope of this thesis.

To determine the performance on both the CPU and GPU, we measure the time it takes for either of the implementations to compute the motion similarity matrix (with kernel convolution) for every possible combination of the 58 sequences used in the classifier using the leave-one-out testing methodology, resulting in 3306 motion similarity matrices of variable sizes to be computed. Table 8.1 lists the final results, computed on the hardware available during the development of this thesis.

A final note on the results, though: The speedup obtained on the GPUs are highly dependent on the GPU architecture. The architecture limits how many of the CUDA cores that can perform double-precision (FP64) calculations, which is used while performing the channel normalization. Unlike the CPUs FPU,

	i5-4590	GTX 750	GTX 760	GTX 780Ti
Total	$1615s$	$578s$	$454s$	$181s$
Speedup	$1x$	$2.79x$	$3.56x$	$8.92x$

Table 8.1: Comparison between an Intel CPU and nVidia GPUs and their computational performance for our motion similarity algorithm. The results show performance increase linearly with GPU CUDA performance.

which performs computations of both single and double precision, the GPU have only a fraction of FP64 units compared to FP32 units that support single-precision floating point operations. Additionally, CPUs have a small number of cores that usually ranges between 4 to 6, while the number of CUDA cores can vary greatly, being 512 on the GTX 750 and 2880 on the GTX 780Ti. The way FP64 CUDA cores translates to computational performance is roughly equivalent to the number of compute-iterations the card must process. If the length of a sequence does not exceed the number of FP64 units available, it will be computed in a single iteration, but exceeding this, several iterations must be done. This greatly affects the speedup achieved, as can be observed of the performance in Table 8.1.

8.5 Summary

This chapter contains a list of the different libraries used to implement our solution, spanning from database interfaces to parallel preprocessor commands and most important, the OpenCV library for image processing. With these libraries, a quicker and more correct implementation is achieved, which allows for more experimentation where needed. Also, porting the motion similarity computation on the GPU allowed for an almost nine times the speedup without any additional optimizations, in other words, a higher speedup is possible.

Another optimization is done on a query that significantly slowed down the server, but a cleaner and faster query was successfully applied. Another implementation detail presented in this chapter, albeit not an optimization, is the coordinate system used throughout all our programs, including both how this coordinate system is laid out and how to translate the relative coordinates to absolute image pixels.

Chapter 9

Conclusion

9.1 Summary and Conclusion

In this thesis, we have proposed a solution for inferring action labels and performing pose estimation in the Bagadus system from video (see Section 1.2.2), to support extensions to the Bagadus system with more analytical capabilities and new entertainment applications. Our proposed solutions solve these two problems by a content-based video retrieval problem. Based on the solution proposed by Efros et al. [15], we developed a prototype workflow of programs that obtains action labels and skeletons in the three procedures of player tracking, video annotation and action retrieval (see Section 1.2.4).

Running the player tracking algorithm in the Bagadus system results a highly precise bounding box around the player, with a near-perfect centering of the torso. Even more, the tracking remains consistent across different types of sequences and actions, with only a few exceptions that does not affect our prototype. A good player tracking algorithm lays the basis for good classification results and directly affects the action classifiaction and skeleton reprojection results.

Using the crowdsourcing platform Microworkers, over 10000 video frames got annotated with skeletons with an unexpected degree of precision (accuracy) from the workers, despite being an unusual scenario for a crowdsourcing campaign. The merged contribution of all the workers were within tolerable margins, making our tool and the crowdworkers effort comparable to an expert. All the frames were annotated in the course of about one week, making the crowdsourcing platform both time and cost effective as well.

A combination of the results of the player tracker and video annotation is used in the action retrieval to obtain the action labels and poses for query sequences in the Bagadus system. By using a similarity measure and classification as proposed by Efros et al. [15], we measure the motion similarity between two sequences using optical flow and find their similarity score. With this score, the best matching annotated sequence, compared to a query sequence, is used to transfer the annotated sequence's action label and skeleton onto the input sequence.

With our proposed prototype, we were able to correctly classify the actions 78% of the time, which is a rather good result given the low resolution and noisy video sequences. Additionally, classes with only one representative class in the annotated database did also get classified correcly, which indicates that this approach can be sufficiently precise (i.e., accurate) to be used in the Bagadus system. The classification algorithm were also ported to the GPU, giving an almost nine times speedup compared to our parallel CPU implementation, with even more potential left in the GPU. Moreover, high accuracy on skeleton reprojections were achieved, with at times pixel-perfect placement of all the joints. The skeletons were obtained using a crowdsourcing platform, and with our filtering techniques and quality assurance mechanisms, we obtained skeletons comparable to an expert's annotations. Although, the skeleton reprojection was not as accurate as desired, but considering that the reprojected skeletons were simply a transfer, shows the viability of this approach. If additional tweaking to the obtained skeletons, or poses, are added, then having pixel-perfect skeletons on every query sequence can be possible.

9.2 Future Work

There is always room for improvements, for example Chapter 5 has a list of remaining issues related to the tracking algorithm and image segmentation. Likewise, the classification algorithm can be further tweaked to improve accuracy and a larger dataset should be used. For the crowdsourcing platform, a proper study of filtering algorithms, task sizes and action classes should increase the reliability of the workers and the final skeletons obtained from them. With the many possible improvements and further developments that can be made, we provide a list of the most relevant and interesting problems for future work, grouped to the three procedures presented in this thesis: player tracking, video annotation and action retrieval.

9.2.1 Player Tracking

Background Subtraction

The accuracy of the player tracking algorithm relies heavily on the precision of the background subtraction algorithm. The algorithm for background subtraction is not perfect and requires a lot of manual tweaking. Improving the parameters and subtraction results will improve the tracker. Likewise, obscured players, camera resolution and animated advertisement should all be isolated out of the tracker, as they all cause tracking failures at times. Because the sequences selected in this thesis were chosen based on action and clarity (i.e., good optical flow results), the tracker is good enough for the prototype system.

Overlapping Players

Another feature to add would be to track and differentiate between overlapping players. Managing to separate and follow the correct soccer player out of two players requires a tracking algorithm that can identify merged foreground mask blobs and be able to tell the difference between overlapping limbs. Because being able to separate players within the same frame is not necessary in order to solve action recognition and pose estimation problems, we have limited ourselves to not include these scenarios in our prototype. However it would be necessary to solve this for a full-scale system.

9.2.2 Video Annotation

Crowdworker Filtering and Merging

The current implementation of filtering and merging of the crowdworker effort is done using Majority Votes. Although this system works with acceptable results, there are still potential for improvements: The most significant improvement would be to resolve the left-right ambiguity that occurs with disagreement among the workers, causing more than one cluster of annotated points to occur in a frame. Currently, when clusters occurs, the merged joint position have a tendency to end up in the middle, which obviously is not correct, as it should be the centroid in one of the clusters. In general, improving the filtering and merging of the crowdworker results, as well as being able to better remove cheating workers, should give better annotated skeletons in the database, which makes crowdsourcing a truly viable option.

Improving Annotation Tools

Improving the filters and merging of all the workers effort can increase the final skeleton accuracy, but increasing the accuracy of individual workers' accuracy is also an improvement that should result in better skeletons. Most notably is the occasions where limbs and joints are ambiguous, causing disagreement and poor final skeletons. Improving this is not required for our prototype, but a necessity for a full-scale system. Further developing the `Online Training Tool` to better support and aid annotating ambiguous and difficult frames should make it easier to use sequences in the database that consist of more complex actions.

9.2.3 Action Retrieval

Improve Accuracy

The accurate results in this thesis have some to be desired (i.e., a higher accuracy would be better for both action classification and skeleton reprojection/pose estimation), including increasing it up from approximately 78 percent and discover why chaining the convolution kernel does not affect the classification result. Both of these problems can be caused by our selection of sequences, which is based on a single isolated player whom performs only common actions. Additionally, testing other classifiers and similarity measures could be interesting for comparisons of methods, but outside the scope for this thesis.

Add Kinematic Constraint

Our implementation is based on Efros et al. [15] and because of this it does not include any skeleton reprojection adjustments nor kinematic constraint checks. Although we do adjust for height differences between sequence reprojections, we do not adjust small misalignment issues. The primary reason for not performing any further skeleton adjustments is because we desired to test the direct reprojection accuracy without additional steps, but as observed, it is a necessity for more accurate results. Moreover, adding kinematic constraints to both crowdworker filtering and reprojections of the skeletons should also increase accuracy is several different parts, but due to the complexity of a kinematic constraint model this means that it is outside the scope of this thesis.

Optimizations

The two prototype implementations provided by this thesis includes a parallel OpenMP implementation and a CUDA GPU port. Both are created to speed up testing and explore the potential for real-time processing. However, because our system is a proof-of-concept prototype, it does not need to run in real-time for the classification. On the other hand, if it were to be implemented as a real-time system, then a great amount of effort must be made into optimizing the source code. In other words, there is a great potential for optimizations on both CPU and GPU code, with the possibility of reaching real-time performance.

9.3 Final Words

Although our system is only a prototype to explore action retrieval and pose estimation in the Bagadus system, it provides exceptionally accurate and robust results, despite the large number of limitations. If the improvements listed in Future Work are added, then there is a good likelihood that our proposed solution can be used with the Bagadus sport analytic system to add more analytic capabilities or other entertainment features.

Index

Bibliography

[1] Najm Alotaibi. ‘Human Motion Capture: New Innovations in the Field of Computer Vision’. In: *International Journal of Computer, Control, Quantum and Information Engineering* 9.3 (2015), pp. 579–582. issn: 1307-6892. url: http://waset.org/Publications?p=99.

[2] Henrik Kjus Alstad. ‘Towards Real-Time Depth Estimation in Large Spaces — A Soccer Case Study’. MA thesis. University of Oslo, 2013.

[3] A. Baak, M. Muller, G. Bharaj, H.-P. Seidel and C. Theobalt. ‘A data-driven approach for real-time full body pose reconstruction from a depth camera’. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. Nov. 2011, pp. 1092–1099. doi: 10.1109/ICCV.2011.6126356.

[4] *Bagadus Trigger Box*. "[Online; accessed 11-Jan-2015]". url: http://mpg.ndlab.net/2013/08/new-trigger-box-design-released/.

[5] D. Batra, Tsuhan Chen and R. Sukthankar. ‘Space-Time Shapelets for Action Recognition’. In: *Motion and video Computing, 2008. WMVC 2008. IEEE Workshop on*. Jan. 2008, pp. 1–6. doi: 10.1109/WMVC.2008.4544051.

[6] M. Blank, L. Gorelick, E. Shechtman, M. Irani and R. Basri. ‘Actions as space-time shapes’. In: *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*. Vol. 2. Oct. 2005, 1395–1402 Vol. 2. doi: 10.1109/ICCV.2005.28.

[7] *Boost C++ Libraries*. url: http://www.boost.org/.

[8] Daren C. Brabham. *Crowdsourcing*. The MIT Press essential knowledge series. The MIT Press, 2013. isbn: 978-0-262-51847-5.

[9] Bryan Catanzaro, Narayanan Sundaram and Kurt Keutzer. ‘Fast Support Vector Machine Training and Classification on Graphics Processors’. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML ’08. Helsinki, Finland: ACM, 2008, pp. 104–111. isbn: 978-1-60558-205-4. doi: 10.1145/1390156.1390170. url: http://doi.acm.org/10.1145/1390156.1390170.

[10] Tat-Jen Cham and J.M. Rehg. ‘A multiple hypothesis approach to figure tracking’. In: *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.* Vol. 2. 1999, pp. 239–245. doi: 10.1109/CVPR.1999.784636.

[11] Chao-I Chen and Roger Stettner. 'Drogue tracking using 3D flash lidar for autonomous aerial refueling'. In: *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics. 2011, 80370Q–80370Q.

[12] German K. M. Cheung, Simon Baker and Takeo Kanade. 'Visual Hull Alignment and Refinement Across Time: A 3D Reconstruction Algorithm Combining Shape-From-Silhouette with Stereo.' In: *CVPR (2)*. IEEE Computer Society, 17th Aug. 2004, pp. 375–382. isbn: 0-7695-1900-8. url: http://dblp.uni-trier.de/db/conf/cvpr/cvpr2003-2.html#CheungBK03a.

[13] Sen-Ching S. Cheung and Chandrika Kamath. *Robust Background Subtraction With Foreground Validation For Urban Traffic Video*. Tech. rep. Center for Applied Scientific Computing Lawrence Livermore National Laboratory 7000 East Avenue, Livermore, CA 94550, 2005.

[14] Peter Denning, Douglas E. Comer, David Gries, Michael C. Mulder, Allen B. Tucker, A. Joe Turner and Paul R. Young. 'Computing As a Discipline: Preliminary Report of the ACM Task Force on the Core of Computer Science'. In: *Proceedings of the Nineteenth SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '88. Atlanta, Georgia, USA: ACM, 1988, pp. 41–41. isbn: 0-89791-256-X. doi: 10.1145/52964.52975. url: http://doi.acm.org/10.1145/52964.52975.

[15] A.A. Efros, A.C. Berg, G. Mori and J. Malik. 'Recognizing action at a distance'. In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. Oct. 2003, 726–733 vol.2. doi: 10.1109/ICCV.2003.1238420.

[16] A. Elgammal and Chan-Su Lee. 'Inferring 3D body pose from silhouettes using activity manifold learning'. In: *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. Vol. 2. June 2004, pp. II-681–II-688. doi: 10.1109/CVPR.2004.1315230.

[17] *FFmpeg*. url: https://www.ffmpeg.org/.

[18] David J. Fleet and Yair Weiss. 'Optical Flow Estimation'. In: *Handbook of Mathematical Models in Computer Vision*. Springer US, 2006, pp. 237–257. doi: 10.1007/0-387-28831-7_15.

[19] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn and H. -P Seidel. 'Motion capture using joint skeleton tracking and surface estimation'. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. June 2009, pp. 1746–1753. doi: 10.1109/CVPR.2009.5206755.

[20] V. Ganapathi, C. Plagemann, D. Koller and S. Thrun. ‘Real time motion capture using a single time-of-flight camera’. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. June 2010, pp. 755–762. doi: 10.1109/CVPR.2010.5540141.

[21] Kristen Grauman, Gregory Shakhnarovich and Trevor Darrell. ‘Inferring 3D Structure with a Statistical Image-Based Shape Model’. In: *In ICCV*. 2003, pp. 641–648.

[22] Jeff Howe. ‘The rise of crowdsourcing’. In: *Wired magazine* 14.6 (2006), pp. 1–4.

[23] Pei-Yun Hsueh, Prem Melville and Vikas Sindhwani. ‘Data Quality from Crowdsourcing: A Study of Annotation Selection Criteria’. In: *Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing*. HLT ’09. Boulder, Colorado: Association for Computational Linguistics, 2009, pp. 27–35. url: http://dl.acm.org/citation.cfm?id=1564131.1564137.

[24] *Human Action Retrieval in the sport analytic system BAGADUS: Source Code*. url: https://bitbucket.org/mpg_code/bagadus-humanactionretrieval.

[25] Texas Instruments. *Introduction to the Time-of-Flight (ToF) System Design*. Tech. rep. Texas Instruments, 2014.

[26] Finn Jacobsen. *Videologi*. Amalie forlag, 2007, pp. 203–255.

[27] *Kinect for Windows Sensor Components and Specifications*. "[Online; accessed 28-June-2015]". url: https://msdn.microsoft.com/en-us/library/jj131033.aspx.

[28] Daniel Kondermann, Steffen Abraham, Gabriel Brostow, Wolfgang Förstner, Stefan Gehrig, Atsushi Imiya, Bernd Jähne, Felix Klose, Marcus Magnor, Helmut Mayer, Rudolf Mester, Tomas Pajdla, Ralf Reulke and Henning Zimmer. ‘On Performance Analysis of Optical Flow Algorithms’. In: *Proceedings of the 15th International Conference on Theoretical Foundations of Computer Vision: Outdoor and Large-scale Real-world Scene Analysis*. Dagstuhl Castle, Germany: Springer-Verlag, 2012, pp. 329–355. isbn: 978-3-642-34090-1. doi: 10.1007/978-3-642-34091-8_15. url: http://dx.doi.org/10.1007/978-3-642-34091-8_15.

[29] Ivan Laptev and Tony Lindeberg. ‘Space-time Interest Points’. In: *Computer Vision, 2003. ICCV 2003. Ninth IEEE International Conference on*. 2003, pp. 432–439.

[30] Matthew Lease. ‘On Quality Control and Machine Learning in Crowdsourcing.’ In: *Human Computation*. 2011.

[31] Babak Loni, Jonathon Hare, Mihai Georgescu, Michael Riegler, Xiaofei Zhu, Mohamed Morchid, Richard Dufour and Martha Larson. 'Getting by with a little help from the crowd: Practical approaches to social image labeling'. In: *Proceedings of the 2014 International ACM Workshop on Crowdsourcing for Multimedia*. ACM. 2014, pp. 69–74.

[32] *MeekroDB*. url: http://www.meekro.com/.

[33] *Microworkers*. url: https://microworkers.com/.

[34] Thomas B Moeslund, Adrian Hilton and Volker Krüger. 'A survey of advances in vision-based human motion capture and analysis'. In: *Computer vision and image understanding* 104.2 (2006), pp. 90–126.

[35] D.D. Morris and J.M. Rehg. 'Singularity analysis for articulated object tracking'. In: *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*. June 1998, pp. 289–296. doi: 10.1109/CVPR.1998.698622.

[36] *Motion Capture in FarCry3*. "[Online; accessed 11-Jan-2015]". url: http://www.fxguide.com/featured/far-cry-3-digital-survivors/.

[37] *Open Multi-Processing*. url: http://openmp.org/wp/.

[38] *Open Source Computer Vision Library*. url: http://opencv.org/.

[39] Svein Arne Pettersen, Dag Johansen, Håvard Johansen, Vegard Berg-Johansen, Vamsidhar Reddy Gaddam, Asgeir Mortensen, Ragnar Langseth, Carsten Griwodz, Håkon Kvale Stensland and Pål Halvorsen. 'Soccer Video and Player Position Dataset'. In: *Proceedings of the 5th ACM Multimedia Systems Conference*. MMSys '14. Singapore, Singapore: ACM, 2014, pp. 18–23. isbn: 978-1-4503-2705-3. doi: 10.1145/2557642.2563677. url: http://doi.acm.org/10.1145/2557642.2563677.

[40] Ronald Poppe. 'Vision-based human motion analysis: An overview'. In: *Computer Vision and Image Understanding* 108.1–2 (2007). Special Issue on Vision for Human-Computer Interaction, pp. 4–18. issn: 1077-3142. doi: http://dx.doi.org/10.1016/j.cviu.2006.10.016. url: http://www.sciencedirect.com/science/article/pii/S1077314206002293.

[41] D. Ramanan and D.A. Forsyth. 'Finding and tracking people from the bottom up'. In: *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*. Vol. 2. June 2003, pp. II-467–II-474. doi: 10.1109/CVPR.2003.1211504.

[42] dr.ir. Ronald Poppe. 'A survey on vision-based human action recognition'. In: *Image and vision computing* 28.6 (June 2010), pp. 976–990. url: http://doc.utwente.nl/70470/.

[43] C. Schuldt, I. Laptev and B. Caputo. 'Recognizing human actions: a local SVM approach'. In: *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*. Vol. 3. Aug. 2004, 32–36 Vol.3. doi: 10.1109/ICPR.2004.1334462.

[44] Thimo Schulze, Dennis Nordheimer and Martin Schader. 'Worker perception of quality assurance mechanisms in crowdsourcing and human computation markets'. In: *Proceedings of the Nineteenth Americas Conference on Information Systems*. 2013.

[45] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman and A. Blake. 'Real-time human pose recognition in parts from single depth images'. In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. June 2011, pp. 1297–1304. doi: 10.1109/CVPR.2011.5995316.

[46] Ricardo da Silva Torres and Alexandre X Falcao. 'Content-Based Image Retrieval: Theory and Applications.' In: *RITA* 13.2 (2006), pp. 161–185.

[47] *Soccer Video and Player Position Dataset*. 2013. url: http://home.ifi.uio.no/paalh/dataset/alfheim/.

[48] *SQL View Performance*. "[Online; accessed 12-June-2015]". url: https://technet.microsoft.com/library/Cc917715.

[49] Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Marius Tennøe, Espen Helgedagsrud, Mikkel Næss, Henrik Kjus Alstad, Asgeir Mortensen, Ragnar Langseth, Sigurd Ljødal, Østein Landsverk, Carsten Griwodz, Pål Halvorsen, Magnus Stenhaug and Dag Johansen. 'Bagadus: An Integrated Real-time System for Soccer Analytics'. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 10.1s (Jan. 2014), 14:1–14:21. issn: 1551-6857. doi: 10.1145/2541011. url: http://doi.acm.org/10.1145/2541011.

[50] E.E. Stone and M. Skubic. 'Silhouette classification using pixel and voxel features for improved elder monitoring in dynamic environments'. In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*. Mar. 2011, pp. 655–661. doi: 10.1109/PERCOMW.2011.5766970.

[51] Hao Su, Jia Deng and Li Fei-Fei. 'Crowdsourcing annotations for visual object detection'. In: *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*. 2012.

[52] *The Revolution in Advanced Sports Analytic Systems*. "[Online; accessed 11-July-2015]". url: http://www.sloansportsconference.com/?p=5503.

[53] Jonathan J Tompson, Arjun Jain, Yann LeCun and Christoph Bregler. 'Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation'. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence and K.Q. Weinberger. Curran Associates, Inc., 2014, pp. 1799–1807. url:

http://papers.nips.cc/paper/5573-joint-training-of-a-convolutional-network-and-a-graphical-model-for-human-pose-estimation.pdf.

[54] Chunyu Wang, Yizhou Wang and A.L. Yuille. 'An Approach to Pose-Based Action Recognition'. In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. June 2013, pp. 915–922. doi: 10.1109/CVPR.2013.123.

[55] Zhou Wang, A.C. Bovik and Ligang Lu. 'Why is image quality assessment so difficult?' In: *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*. Vol. 4. May 2002, pp. IV-3313–IV-3316. doi: 10.1109/ICASSP.2002.5745362.

[56] "*Workflow, Pipeline*" *Oxford English Dictionary 4th Edition*. Oxford University Press. Semptember 2014. url: http://dictionary.oed.com/.

[57] Alper Yilmaz, Omar Javed and Mubarak Shah. 'Object Tracking: A Survey'. In: *ACM Comput. Surv.* 38.4 (Dec. 2006), p. 13. issn: 0360-0300. doi: 10.1145/1177352.1177355. url: http://doi.acm.org/10.1145/1177352.1177355.

[58] Omar F. Zaidan and Chris Callison-Burch. 'Crowdsourcing Translation: Professional Quality from Non-professionals'. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. HLT '11. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 1220–1229. isbn: 978-1-932432-87-9. url: http://dl.acm.org/citation.cfm?id=2002472.2002626.

[59] Zheng Zhang, Hock-Soon Seah, Chee Kwang Quah and Jixiang Sun. 'GPU-Accelerated Real-Time Tracking of Full-Body Motion With Multi-Layer Search'. In: *Multimedia, IEEE Transactions on* 15.1 (Jan. 2013), pp. 106–119. issn: 1520-9210. doi: 10.1109/TMM.2012.2225040.

[60] Yuxiang Zhao and Qinghua Zhu. 'Evaluation on Crowdsourcing Research: Current Status and Future Direction'. In: *Information Systems Frontiers* 16.3 (July 2014), pp. 417–434. issn: 1387-3326. doi: 10.1007/s10796-012-9350-4. url: http://dx.doi.org/10.1007/s10796-012-9350-4.

[61] *ZXY Sport Tracking*. url: http://www.zxy.no/.