BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

RUNNING MOTION ANALYSIS

SYSTÉM PRO ANALÝZU POHYBŮ PŘI BĚHU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **RADOSLAV ELIÁŠ**
AUTOR PRÁCE

SUPERVISOR **Ing. TOMÁŠ GOLDMANN,**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2020/2021

Bachelor's Thesis Specification

Student: **Eliáš Radoslav**
Programme: Information Technology
Title: **Running Motion Analysis**
Category: Artificial Intelligence

Assignment:

1. Get acquainted with existing tools for running motion analysis. More importantly, focus on solutions that use cameras.
2. Study available methods and algorithms of computer vision for human movement analysis. Mainly get familiar with algorithms for skeleton detection.
3. Design a system with two cameras for capture human movement. The first camera should be record the person from the back and the second camera from the side.
4. Implement an application that should be capable merge data from both cameras and extract trajectories of movements of leg joints (knee and ankle).
5. Test the solution on your videos. Discuss the possibilities of using the system.

Recommended literature:

- MORAIS, Romero, et al. Learning regularity in skeleton trajectories for anomaly detection in videos. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019. p. 11996-12004.
- PIPKIN, Andrew, et al. Reliability of a qualitative video analysis for running. *journal of orthopaedic & sports physical therapy*, 2016, 46.7: 556-561.

Requirements for the first semester:

- Items 1 a 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Goldmann Tomáš, Ing.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2020
Submission deadline: May 12, 2021
Approval date: April 9, 2021

Abstract

The goal of this thesis is to analyze body movement in running gait. The system works with recordings from two cameras, one from the side and one from the back. The problem is solved using a pose estimation algorithm based on the convolutional method. Multiple estimators are compared in this thesis. The final system uses the OpenPose framework and provides a library with calculations for many metrics used to evaluate the running gait. Results are then visualised in a multiplatform desktop application. Experiments were conducted on a private dataset of running recordings.

Abstrakt

Cieľom tejto práce je analyzovať pohyb a držanie tela pri behu. Systém pracuje so záznamom z dvoch kamier, zboku a zozadu. Využíva nástroj na detekciu postoja ľudského tela založenú na konvolučnej metóde. Práca porovnáva niekoľko detektorov. Výsledný systém používa detektor OpenPose a implementuje knižnicu s výpočtami pre rôzne metriky používane na ohodnotenie formy behu. Výsledky sú zobrazené v multiplatformnej aplikácii. Ohodnotená bola niekoľkými experimentmi na osobnej dátovej sade videí behu.

Keywords

artificial intelligence, neural networks, computer vision, skeleton detection, pose estimation, running form, running, running gait, AI, movement detection, body position, camera, biomechanics, Python, kinematics, video

Klíčová slova

umelá inteligencia, neurónové siete, počítačové videnie, detekcia skeletu, detekcia postoja, bežecká forma, beh, držanie tela, detekcia pohybu, pozícia tela, kamera, biomechanika, Python, kinematika, video

Reference

ELIÁŠ, Radoslav. *Running Motion Analysis.* Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Tomáš Goldmann,

Running Motion Analysis

Declaration

Hereby I declare that this bachelor's thesis was created solely by me, under the supervision of Mr. Ing. Tomáš Goldmann. Every source and publication used in this thesis is properly cited and included in the list of references.

.....
Radoslav Eliáš
May 3, 2021

Acknowledgements

I would like to thank my supervisor Mr. Ing. Tomáš Goldmann for his personal approach and willingness to help with any problems during the work on this thesis.

Contents

Chapter 1

Introduction

Sports and exercise are an essential part of staying healthy and keeping our bodies working properly. According to the World Health Organization, one in four people is physically inactive, causing disease and shortening life expectancy up to 10 years [26].

Introducing people to exercise is now more important than ever. Walking and running could be considered as some of the simplest, yet most efficient activities available for beginners. Most people already have all they need to start running, but with the vast availability of this sport comes the problem of proper technique and safety. Bad form can inflict unnecessary knee pain and many more injuries, caused by too much pressure being applied to joints, ligaments, and tendons.

While medicine and mainly physiotherapists can fix these issues, the average beginner runner usually does not bother seeking professional help until it is too late and some damage has already been done. The goal of this paper is to automatize this process by analyzing video recordings and make it accessible to anyone. With smartphone cameras improving every year, a system like this will enable any person to record himself and adjust his running form, which will eventually lead to injury prevention.

Chapter 2 focuses on biomechanics as a study and further dives into running, gait, and body position. It talks about the essential parts of a good running posture and how they help prevent injuries and other health problems. The next Chapter 3 presents the basics of neural networks, standard methods of computer vision, and how they are used to detect and track human poses from a video. Afterwards, the main focus is joint extraction and position comparison with other body parts. Propositions for the final system are discussed in Chapter 4, along with the implementation details for the individual parts. Experiments done on the resulting product are explained in Chapter 5.

Personally, this topic is very important to me as a passionate runner, which is why I want to help other people discover the beauty of my sport by minimizing the risks involved.

Chapter 2

Human biomechanics and running analysis

2.1 Biomechanics

Biomechanics is the study of the structure, function, and motion of the mechanical aspects of biological systems [2]. It basically means that this field studies how all animals and humans move, how their bodies function, and why we are able to just stand up from our desks and walk to the nearest coffee shop. The ability to do that is called the **locomotive function**. This chapter will focus specifically on human biomechanics. We, as a species, have a big range of movements and postures that we are able to perform. All these functions are part of human biomechanics.

The main real-world application of this study is physiotherapy, which aims to improve physical performance and prevent or recover from movement-related injuries.

Terminology

Parts of this section are paraphrased from [16].

Human biomechanics studies forces affecting human muscles and bones and how the body tissues respond to these forces. They can be divided into 2 different groups, internal and external. External forces are caused by an outside environment, for example, the ground pushing on a foot as a person takes a step. Most of them are contact forces, which means that the object applying the force is in contact with or touching the human body. Internal forces are generated by the body itself, therefore the name. This paper will mostly work with external ones.

Furthermore, biomechanics are split into static and dynamic concepts. Considering the topic being running motion, we will talk about the dynamics. They study the conditions under which an object moves.
There are 2 types of dynamic concepts:

- Kinematics
- Kinetics

Simplified kinematics study and describe the motion of an object without referencing the forces causing the said motion, e.g., a football rolling from place A to place B. In contrast, kinetics take those forces into consideration, e.g., a player kicking the football causing that movement.

Kinematics

There are five variables studied in kinematics [1]:

- Type
- Location
- Direction
- Magnitude
- Rate

Type of motion can either be linear, which means all body parts are moving in the same direction along a straight line, or angular. Angular motion is in other words, rotation around axis.

Location of a joint is usually defined in a system of anatomical planes and axes. There are three planes of motion. Sagittal, transverse (sometimes referred to as horizontal), and frontal. Shown in 2.1.

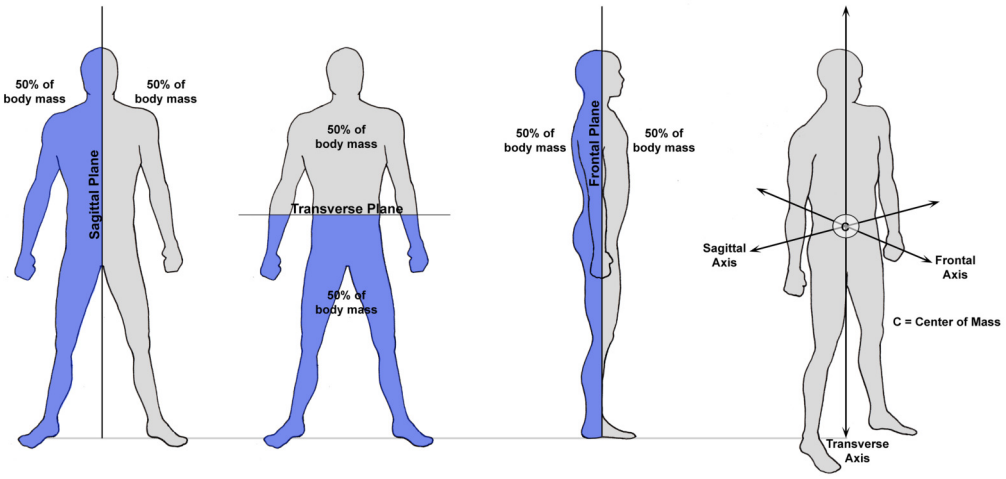


Figure 2.1: Three planes of motion, source:[9]

Direction of motion is also described by the three basic planes of motion. Flexion and extension are the two opposing directions in the sagittal plane. Flexion motion decreases the angle between two body parts while extension increases them. When a motion reaches an extreme on one side of the range, it is often referred to as *hyper*. In the frontal plane, the main movement is called abduction, which goes further away from the midline of the plane. Its opposite is adduction, in other words, moving back towards the midline. The most common motions in the transverse plane are internal and external rotations.

For linear motions, **magnitude** represents the distance between two spots that the object traveled. Magnitude of angular motion is measured in radians or degrees.

Rate of motion is in other words, the speed or velocity of the studied object. Sometimes acceleration is also one of the variables looked at.[10]

Kinetics

As stated before, kinetics studies the forces that cause the motion of an object. Force can be described as an act of one object on another and can be either internal or external. To determine how will the object or a human body, in this case, react, all existing forces must be taken into consideration or the result will not be precise.

Well known Newton's laws also describe the kinetics of an object. Law of inertia or in other words Newton's first law states that the higher the mass, the stronger the force needed to transpose an object from a standstill to a moving motion. Newton's second law shows that a number of forces will affect an object to increase or decrease its velocity depending on the direction of the forces. The third law states that for every action there is an equal reaction. A simple example in running is that a runner will run faster on a concrete road than on sand with the same effort because the concrete will create a bigger reaction or „pushback".[16]

Application

Biomechanics essentially study the movement technique, which is most often applied in sports. The goal is usually either one of two things. Either to prevent injuries or at least reduce the risk of them or to improve performance by adjusting or removing unnecessary movements. There are a few different areas where biomechanics are applied [12]:

- The identification of optimal technique for enhancing performance
- The assessment of muscular recruitment in order to prevent overloading
- The analysis of body loading to determine the safest method of performing exercise
- The analysis of sports equipment e.g., shoes, bicycles, rackets.

2.2 Biomechanics in sports

Center of Gravity

Center of gravity or COG for short, is an imaginary point around which body weight is evenly distributed. This concept is important to understand stability, balance, and their effects in sports with rapid body movement. The Base of support is an area beneath an object which includes every contact point of the said object and a surface. COG changes often and when the line of gravity falls outside the base of the support, adjustment is needed to stay balanced. [10]

Balance

Balance is the ability of a person to control his or her stability. Simplified, it means keeping the body under control and moving only where the person wants to.

Static Balance

We talk about static balance when the body is stationary. It is the ability to maintain a fixed posture while at standstill.

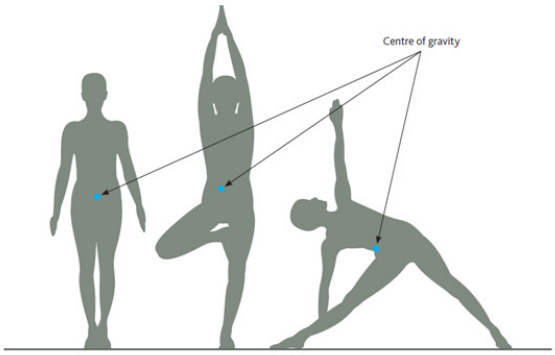


Figure 2.2: Center of Gravity, source: [25]

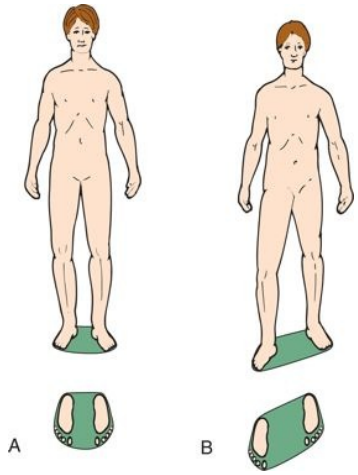


Figure 2.3: Base of Support, source: [16]

Dynamic Balance

Dynamic balance is much more demanding and problematic. Developing babies learn this later than static balance. It consists of keeping the center of mass over the base of the support at all times while constantly changing the body position and with it, the center of gravity. Athletes like gymnasts need to perfect this skill to perform in their field.

Momentum

Momentum is the product of an object's velocity and mass and is closely linked to kinematics. In short, it is the amount of motion. Momentum can be transferred between objects [10]. There are 2 types of momentum.

Linear Momentum

As the name suggests, the linear momentum is in a straight line like running, walking, or cycling in one direction.

Angular Momentum

Angular or in other words, the rotational momentum is created by the said rotations of objects. A good example of this is tennis and how good players often do not hit the ball perpendicular to a racket but with a slight angle which creates more power.

2.3 Running analysis

Running, while similar to walking, has a few key differences crucial to understanding proper technique and injury prevention. Each person has different flexibility, strength, and body composition. When determining a proper running posture, it is important to take all these variables into a consideration. Fatigue plays a big role too because it is harder to focus on

body position when muscles are getting tired. Here are a few of the most common problem areas contributing to the incorrect running technique [13].

Feet

Feet are the only contact point with the ground while running. That means they take the biggest abuse and are very prone to damage and injuries. When running, a person should hit the floor with the ball of the foot. The two most common problems are striking with toes or with the heel. These may not be harmful after one or two runs, but in the long term can cause a lot of pain. A less common but still existing type of bad foot placement is hitting the ground at an angle - both the left and right sides of the foot should hit the surface at the same time. Another incorrect gait type is when the front of a foot points slightly either outwards or inwards. It should always follow the direction of the movement of a person in a straight line.

Legs and Knees

Effects of legs and knees are closely related to the foot position discussed earlier. When running, it is important to move the legs in a straight line, the same as with feet. Every stride should be done with the shin being perpendicular to the surface, or at least as close to perpendicular as possible. Inconsistence in this particular position causes the aforementioned heel or toe striking. When the angle of the shin is too big, the heel drops first and hits the ground before the rest of the foot. Otherwise, if the angle is too small, the toes might get damaged by the repeated load.

Torso

Torso makes up the biggest mass of the human body, so using it properly can greatly increase one's performance. The Center of gravity is also situated here while running, meaning that even a slight movement will cause big changes in effectiveness. Again, common problems are leaning too far forward or too far backward. Even though the torso should be fairly straight, a slight lean forward will drive the body in that direction. However, when it is overdone, the hips have a limited range of motion, and the bigger the lean forwards, the more of that range we lose. On the opposite end, when leaning backward, the body needs more power to actually move in the wanted direction. Still, the torso should not be totally static while running, but should slightly rotate with every stride.

Shoulders and Arms

The upper body is usually not the part people think about when talking about running, but a simple test of sprinting with straight arms attached to one's torso shows how much speed actually comes from this part of the body. Similarly to the torso, shoulders and arms should move with every stride but in a sort of „X“ pattern. Meaning that with the left foot forward, the right shoulder and arm should come forward, and the other way around. Shoulders should be relaxed as not to waste energy but slightly pulled back, almost as squeezing a pencil between the shoulder blades. Arms should be close to the torso, bent at a 90-degree angle at the elbow. Moving them slightly up and down with every stride helps with driving the body forward, almost as pushing itself in the direction it is headed.

Head

The head is the brain of the body, clearly. We naturally move in the direction our eyes are looking. That is why looking at our feet may cause the torso to lean forward, the shoulders hunch over, and so on. Focused gaze makes or breaks proper posture while running. Ears should be aligned with the shoulders.

2.4 Running gait cycle

The running gait cycle can be described as a series of movements that combined, create one repetition of a running motion. It starts with one foot hitting the ground, goes through launching forward from that position and striking with the other foot, and finishes where it started, hitting the surface with the first foot. This is called one gait cycle and repeating it creates the running motion. There are three key positions in the gait cycle: Initial contact, mid stance, and mid flight. Check 2.4 for visual representation.

Form analysis in clinical environment

Analyzing running form and posture most commonly occurs in a physical therapist's clinic. There are two different approaches to this.

The first is when the runner is jogging on a treadmill and the physical therapist observes in real-time. The advantage of this type of analysis is the immediate actions that can be taken. Small adjustments in the technique can be observed in the same session, thus the ability to experiment with different problem fixes, etc.

The other approach is when the running is recorded with a camera and the video is then analyzed. This approach also creates advantages, for example, the possibility of multiple therapists assessing the running technique, without the need of them being at one place at the same time. The video can be sent by the patient even without visiting the clinic, although more common is recording the video there. The main reason for this is the importance of a precise camera set up to ensure good visibility of the body parts of interest [21]. Running recording is usually performed on a treadmill, in order to eliminate surface inconsistencies like potholes and such. Two cameras are used, more commonly, one from the side and one from either the front or back of the runner. Each view shows different aspects of running posture, which is the reason one would be incomplete without the other. For example, the frontal view will not show anything about torso lean or knee flexion angle [17]. Higher framerates are desirable in cameras when recording running because they enable slowing down the video and examining it frame by frame.

In 2016, a study [21] was made by the University of California in San Francisco with an objective to create a framework for systematic video-based running analysis. Testing and analysis were performed on athletes with confirmed running-related injuries.

The runners were recorded while running on a treadmill after a warmup with two cameras. One from the side and one from the posterior view. Colorful markers were applied to some parts of their clothing to catch movements that would otherwise be hidden by the wardrobe.

Variables of interest like angles or distance between body parts were then identified visually on a slow-motion video. Each position of the running gait cycle is used for different

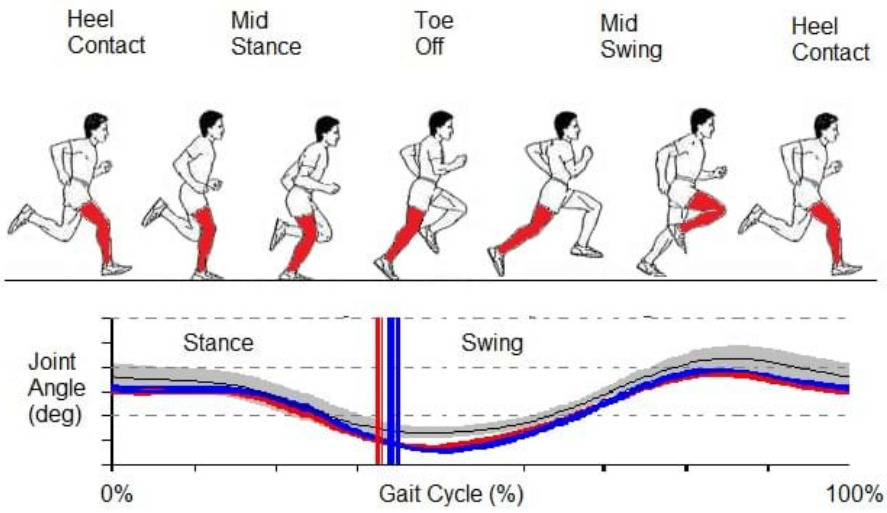


Figure 2.4: Running gait cycle, source:[7]

kinematic variables and correct identification of these moments is crucial for an accurate evaluation of the running gait.

Side view

From the side view, these variables were studied:

- Foot strike pattern 2.7
- Foot inclination angle at initial contact
- Tibia angle at loading response
- Knee flexion during stance
- Trunk lean
- Overstriding 2.8
- Cadence

Proper technique in most of these was discussed in 2.3. Overstriding is the result of combining multiple of these variables causing footstrikes far away from the base of support. The correct cadence is still unclear to scientists, but the estimations are around 180 steps per minute. Lower cadence can lead to overstriding and health problems that come with it.

Posterior view

- Base of support
- Heel eversion
- Heel whips

- Knee window 2.6
- Pelvic drop 2.5

The posterior view mainly focuses on heel positioning and movement with a few other variables. The Knee window is the space between knees. Ideally, the window should be as small as possible while the knees never touch. Pelvic drop is identified in the stance phase by the markers applied to the runner's body. It is the position difference between the marker on the stance leg and the marker on the swing leg demonstrated in 2.5. An excessive pelvic drop may lead to injury.

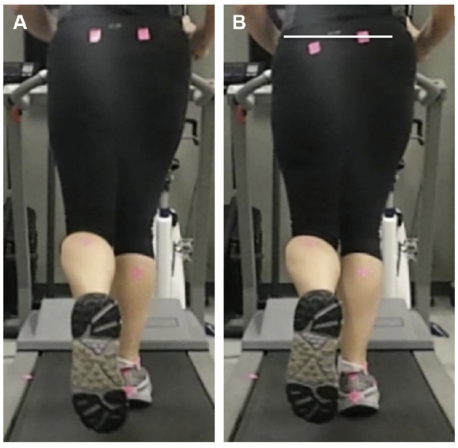


Figure 2.5: [A] Normal pelvic position, [B] excessive pelvic drop, source: [21]

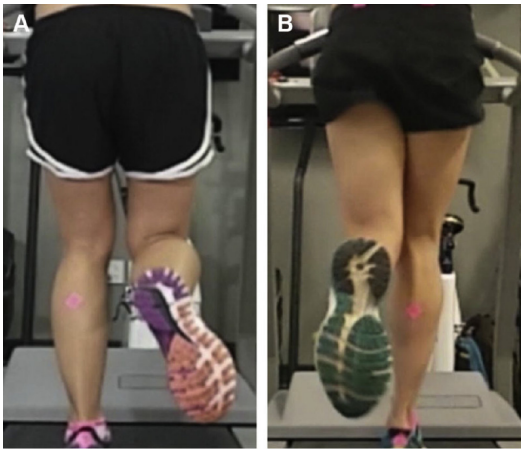


Figure 2.6: [A] Normal knee window, [B] „closed“ knee window, source: [21]

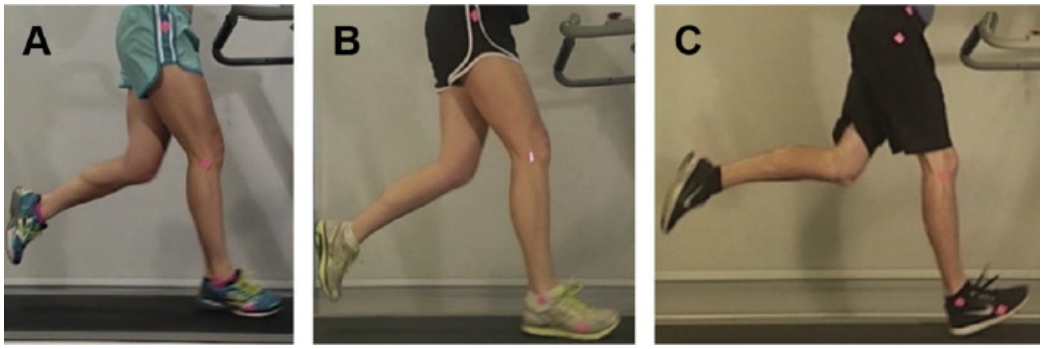


Figure 2.7: [A] Forefoot strike, [B] midfoot strike, [C] rear foot strike, source: [21]

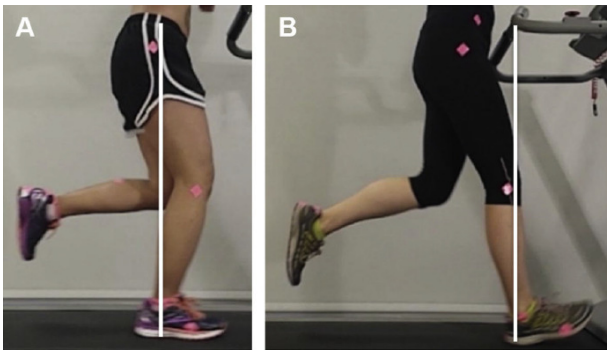


Figure 2.8: [A] Normal stride demonstration, [B] overstriding, source: [21]

Chapter 3

Human pose detection from video

The scientific field of **Computer vision** deals with the problem of detecting objects from a static image or a video recording. Computer vision is a subset of a much larger field, the **Artificial intelligence** . The goal is to understand and analyze digital images from the real world in order to create numerical data that make sense to computer processors. The automatization of the complex abilities that the human visual system has opens up many possibilities. Common problems solved using computer vision are scene reconstruction, video tracking, object recognition, image restoration, motion estimation, and others [27].

3.1 Neural Networks

Neural networks or NNs for short are the standard systems used in artificial intelligence. They are massive, parallel computing systems consisting of a large number of intertwined simple processes. The name and the inspiration behind these are the biological neural networks that make up the animal brain. Neural networks have the ability to learn, which makes them an ideal tool for analyzing unknown inputs. Information in this section is taken from [11].

Biological neuron

A neuron, in the biological context, is a cell with the ability to process information. A single neuron consists of multiple parts, each with its signature role and function. These are shown in Figure 3.1:

Neuron accepts a signal from other neurons, processes data from the input, and creates a new modified signal which is then forwarded to other neurons.

The nucleus works as a storage for information about the neuron's traits. Dendrites are receivers for all incoming signals and the axon is a transmitter for the final signal generated by the cell's body. Axon at the end branch into multiple strands. The place of contact between one neuron's dendrite and another neuron's axon strand is called a synapse.

Chemicals called neurotransmitters are released when an input reaches a synapse. These can adjust the effectiveness of a neuron, enabling humans to learn and remember history.

Every neuron is connected with about $10^3 - 10^4$ other neurons. The frequency of communication between units is only a few hundred Hertz, meaning it is much slower than a modern computer processor. And yet, the human brain performs complex tasks like face recognition in just a few milliseconds. In conclusion, the brain runs a parallel program about 100 steps long.

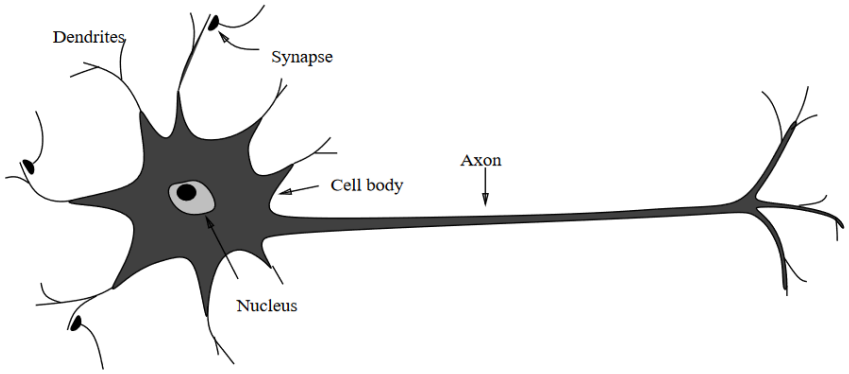


Figure 3.1: Biological neuron, source:[11]

Computational model

The first computation model of a neuron was created by two scientists called McCulloch and Pitts. This model computed the weighted sum of multiple inputs and compared it to a threshold value. The output was either „1“ or „0“, depending on the comparison with the threshold. This model has a few limitations and differences with the biological neuron. For example, using threshold values instead of graded responses and the lack of asynchronous updates of the neuron [11]. It was later generalized into a model called **perceptron**, shown in 3.2.

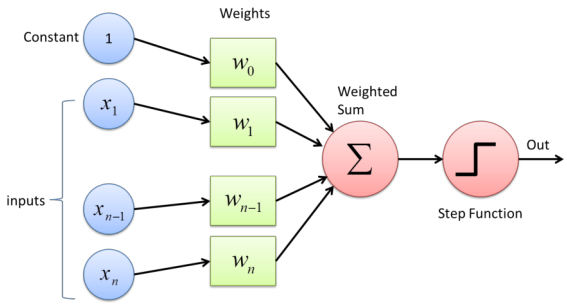


Figure 3.2: Perceptron - computation model of a neuron, source:[20]

Correlations between the two models are:

- Synapses = weighted sums
- Dendrites = input wires
- Axons = output wires
- Cell body = activation function

The mathematical model of a neural network is as follows.

$$y = f(b + \sum_{i=1}^{n} w_i x_i)$$

Where **b** is the bias, **n** is the number of inputs, and **w** are the weights of the corresponding inputs, and **f** is the activation function.

Activation function

Step function or activation function determines the output of a single neuron. These can be split into two categories, linear and nonlinear. The activation function maps the output value in some specified range, depending on the type of the function. Most commonly used step functions are listed in 3.3.

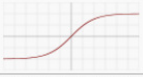
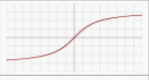
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figure 3.3: Commonly used activation functions, source:[19]

Weight initialization

Weight initialization is the process of determining weight values and biases before the training phase of the network. Correct initialization has a big effect on how fast the network can learn. Bad starting weights can lead to never reaching the required accuracy of the neural network [28].

With zero knowledge of the final system, there are two possible techniques of initialization. Either setting all biases and weights to zeroes or random numbers. With zero initialization, the biases have no effect on the outcome. Zero value weights, on the other hand, will cause symmetric relations among units and therefore the weights will be exactly the same in all other iterations.

Random initialization solves this problem, which is why it is used more commonly. However, with it comes the issue often referred to as the **vanishing gradient**. When the

starting weights are initialized at very high values, the activation function maps them near the value „1“. This causes the weights to change very slowly and ultimately the learning takes a lot of time. Equally, the same phenomenon happens with weights initialized with low values mapping them close to „0“ [28].

To avoid this issue, the initialization technique needs to be compatible with the activation function of the neural network. There are two commonly used methods nowadays, he technique and Xavier initialization.

He method is used with the **ReLu** and **leaky ReLu** activation functions. First, random numbers are generated for each weight and then every value is multiplied by $\sqrt{\frac{2}{n_{in}}}$, where n_{in} is the number of input signals.

Xavier technique is a modification of the previous initialization. The procedure is the same, but with the multiplication value being either $\sqrt{\frac{1}{n_{in}}}$ or $\sqrt{\frac{1}{n_{in}+n_{out}}}$, where n_{out} is the number of output signals.

Neural Network Learning

Information in this section is paraphrased from [11]. The learning process of a neural network can be described as a process of updating the architecture of a network to perform a specific task efficiently and precisely. The weights of the inputs are changed iteratively according to the data fed to the network. The ability to learn is the key feature of how we are able to simulate the human brain.

Unsupervised training

The simplest way of training a network is called unsupervised. The data are fed into the network without any hints about which input pattern is correct. A network trained this way will split the data set into groups with similar characteristics. The number of output chunks can be set before the training.

Supervised training

In supervised training, every input pattern carries information whether it is correct or not. The network calculates the output for a pattern and compares it with the expected result. How much these two differ dictates the error of the network. The weights of the network are then altered according to the error to match the correct output as closely as possible. This process is repeated until the error value is lower than the predefined deviation. This is also often referred to as learning with a teacher.

Convolutional neural networks

A convolutional neural network or CNN is an algorithm used to analyze images, assign weights and biases to any objects in the image, and distinguish among them. These networks consist of multiple interconnected **layers**. Usually one input layer, one output layer, and a few hidden layers in between. Thanks to their architecture, the accuracy is not affected by position, rotation, and other transformations of the objects. This algorithm is again inspired by the human brain and its visual cortex [18].

Convolution layer

The function of this layer is to detect local features in an image using a filter also called **kernel** [29]. The dimensions of the final feature are determined by three parameters:

- **Stride** tells us how much the kernel moves with every step of the method. For a stride equal to one, the filter moves by one pixel, and so forth.
- **Padding** is sometimes added to the original image to gain more information about the edges of the input image. Thanks to the additional borders added by padding, the edge pixels will be used more times in the convolution process.
- **Depth** equals the number of kernels we apply in this layer. For example, in generic RGB images, each pixel has three values corresponding to each color profile, so we might use a different filter for each.

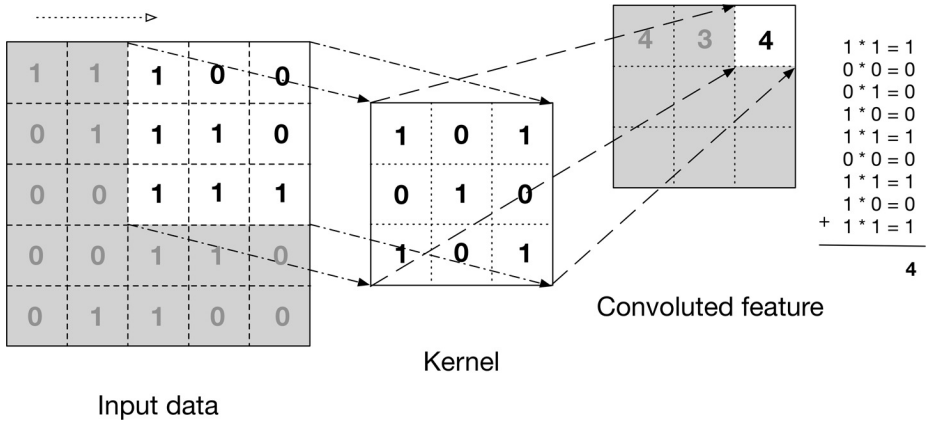


Figure 3.4: Demonstration of a convolution layer, source:[29]

Pooling layer

This layer is often used before the convolution layer. Its purpose is to lower the dimensions of the input data. This is done to decrease the computational requirements of the final network. Multiple pixels are converted into one according to the rules of pooling. Two common types are **max-pooling** and **average pooling**. In max-pooling, only the highest value is kept. Average-pooling calculates the average value of input fields creating the final value [18].

Fully connected layer

A layer of this type has each input node connected to each output node. It means they are costly in terms of computational power needed, but also very precise. Fully connected layers are often used in the last layer of CNN and used for object classification.

3.2 Human pose estimation and tracking

Human pose estimation is a rapidly evolving field and a focus of interest of many researchers. It is characterized as the problem of identifying human joints like elbows, knees, etc. Usu-

ally, in the context of stale images, it's called pose **estimation** and we talk about pose **tracking** if it's in a video recording. 2D pose estimation results in (x,y) coordinates for each joint, while 3D estimation also approximates the **z** coordinate. Example of 2D pose estimation is show in 3.5. The challenges of this task are poor image or video quality, bad camera angle resulting in low visibility of some joints, and baggy clothing hiding the true body pose [4].

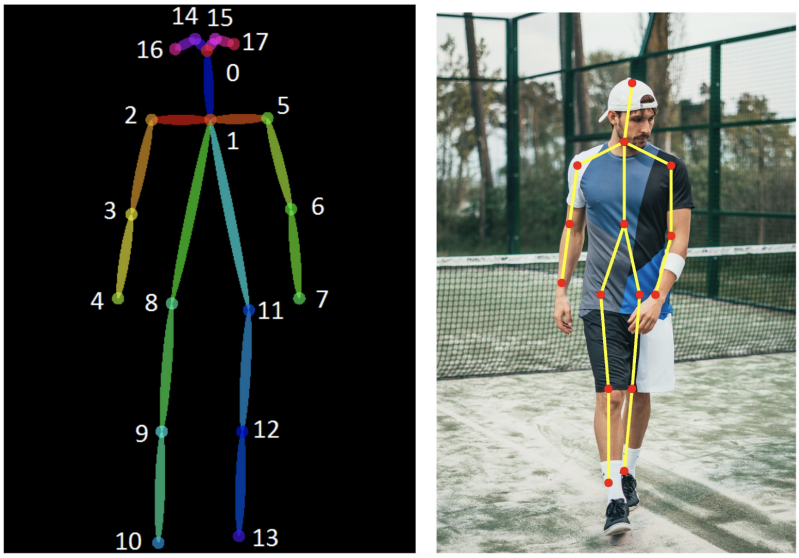


Figure 3.5: Visualisation of 2D human pose estimation algorithm, source:[4]

Nowadays, the most recent human pose estimation algorithms use convolutional neural networks as their core building block [4]. A notable framework that revolutionalized this field was DeepPose [23] by Google, which was the first major paper that applied deep neural network learning to pose estimation.

DeepPose

The DeepPose model solves the problem with a cascade of regressors which transform the input image into a normalized pose vector [23]. It means that after the first joint estimation from the original image, the same algorithm is applied to the much smaller area around the estimated position in the first iteration. This model works with a fixed input size of 220×220. The main disadvantage of this approach is the substantial computational power required for each iteration of the cascade.

By the authors' testing and evaluation, after a single regressor, the joint location estimation is much worse than any of the state-of-the-art approaches available at that time. However, after two or three cascading stages, the accuracy is better than theirs.

Method classification for multi-object human pose tracking

Top-down approach

The top-down approach in human pose tracking is performed in two steps. First, a detection module identifies human objects, and then, a pose estimation algorithm is applied to locate joints and other keypoints [14]. The advantage of this method is that the problem is divided

into separated, smaller, and easier tasks. Additionally, many detection modules usable in this scenario already exist.

Bottom-up approach

Bottom-up, on the other hand, detects all human joints and key points in the frame using a pose estimator. After that, the joints are assembled into individual people according to some data association technique [14]. This approach has an excellent computational cost but sometimes lacks accuracy, depending on the quality of the human assembly technique.

Object tracking

In the past, object detection was accomplished by regressing the input image into bounding box coordinates, an example of this method is shown in 3.6. The object, in our case the human, is then expected inside the bounding box in the next frame. In human pose estimation, this is usually done by regressing the image into heatmaps, where each channel represents a human joint [14]. The heatmap approximates the joint location in each pixel. These heatmaps can then be used to create bounding boxes for each tracked human. Heatmap technique is demonstrated in 3.7.

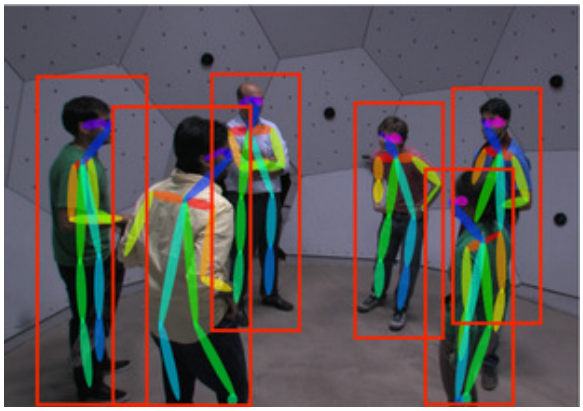


Figure 3.6: Example of object tracking by the bounding box approach, source:[6]

Figure 3.7: Example of the heatmap application for joint estimation, source:[4]

Common pose estimation metrics

Information in this section was taken from [4]. There are a few metrics commonly used to evaluate the performance of human pose estimation frameworks. They are used to compare the models to choose the best one for our application.

Percentage of correct parts (PCP)

This metric measures the detection rate of arms and legs. Distance between the limb joint estimated location and its actual location is calculated. If this distance is smaller than half of the length of the limb, the limb is considered detected. This is often denoted as PCP@0.5.

Percentage of detected joints (PDJ)

The same distance between the estimated joint position and the actual position is measured as before. The joint is referred to as detected if the value is within a certain fraction of the torso diameter [24]. Torso diameter is sometimes switched to the bounding box diagonal, illustrated in 3.8. This is done in order to eliminate problems with people turned sideways which appear to have zero torso diameter between shoulders [22]. The mathematical calculation is as follows [22].

$$PDJ = \frac{\sum_{i=1}^{n} bool(d_i < 0.05 * diagonal)}{n}$$

Where

- n - the number of keypoints in the image
- d_i - the euclidian distance between the true location and estimated location
- bool(condition) - a function that returns one or zero based on the condition

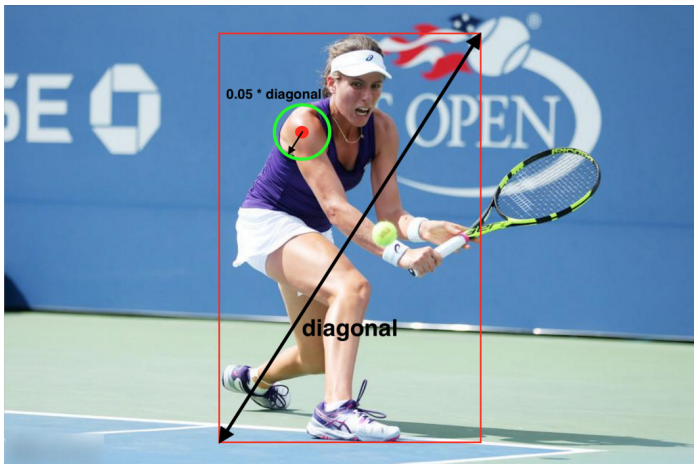


Figure 3.8: Illustration of bounding box diagonal approach to PDJ, source:[22]

Object keypoint similarity (OKS)

Object keypoint similarity is calculated by this equation [22]:

$$OKS = exp(-\frac{d_i^2}{2s^2k_i^2})$$

Where

- d_i - the euclidian distance between true location and estimated location
- s - scale or the square root of the object segment area
- k - per-keypoint coefficient that controls the fall off

Not all keypoints or joints in this case have the same size, that is, what the scale and keypoint coefficient are for. For example, the knee location and ear location are not equal. The distance is normalized by these two variables, they determine the importance of each keypoint [22]. Constants for human joints calculated by [8] are listed and visualised in 3.9.

Keypoint	k_i
hips	0.107
ankles	0.089
knees	0.087
shoulders	0.079
elbows	0.072
wrists	0.062
ears	0.035
nose	0.026
eyes	0.025

Figure 3.9: Keypoint constants for OKS metric, source:[8]

OKS indicates how close the estimated position is to the true one. It is a number between 0 and 1. The bigger, the better. All research papers use this metric with an addition of a threshold either 0.5 or 0.75 [22]. If OKS is greater than the threshold, the keypoint is considered detected. Commonly shown metric is **mAP** or mean Average Precision. It is the mean value of all OKS values.

Pose tracking benchmark

In 2017, a benchmark with a corresponding dataset for pose tracking was created by scientists at Cornell University [3]. This benchmark was published by a community-driven webpage called „Papers with Code“ [15]. Many state-of-the-art up and coming frameworks for pose detection were submitted for testing. The results were then sorted by two variables: **MOTA** or Multiple objects tracking accuracy and **mAP** or mean average precision. Frameworks with the highest MOTA are illustrated in 3.10.

For running motion analysis, MOTA is not as important because there is usually only one moving object in a running video.

LightTrack

LightTrack is a generic framework for 3D human pose tracking created by researchers at the University of Pittsburgh [14]. It was first introduced in 2017 and is being kept updated to this date. This framework utilizes the top-down approach explained in 3.2 to stay very light-weight and still keep competitive performance with other state-of-the-art solutions. LightTrack is capable of both offline and online tracking with comparable accuracy. There are two main modules incorporated into this tool. A **single-person pose tracking** module

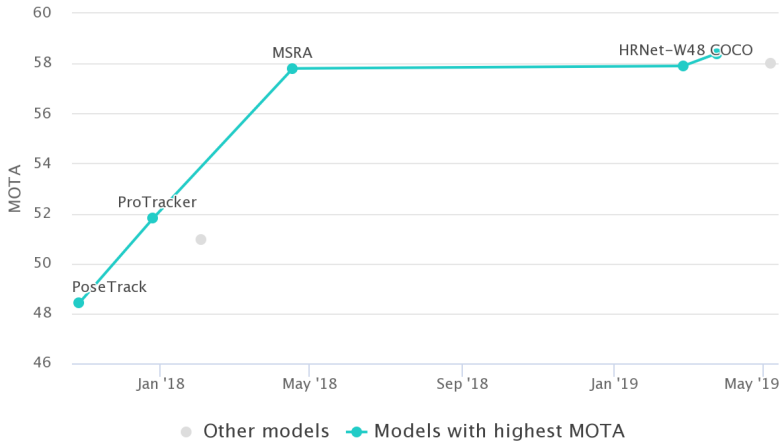


Figure 3.10: Graph showing MOTA performance of frameworks on the PoseTrack2017 benchmark, source:[15]

and a **visual object tracking** one, which combined perform the pose tracking. Overview of this pipeline is presented in 3.11. The authors of this framework aimed to create a generic enough codebase to enable the replaceability of both the human pose estimator and re-identification (Re-ID) modules. They also introduced the first of a kind Re-ID module based on a Siamese Graph Convolutional Network. This module, contrary to others, uses a spatial graph representation of a human skeleton to reidentify lost objects in a video. This approach is computationally inexpensive and robust to sudden camera shifts like those in a football live broadcast. A commonly used bounding box solution for object tracking is also presented here. This area is enlarged by 20% on each side and used to localize the tracked object in the next frame.

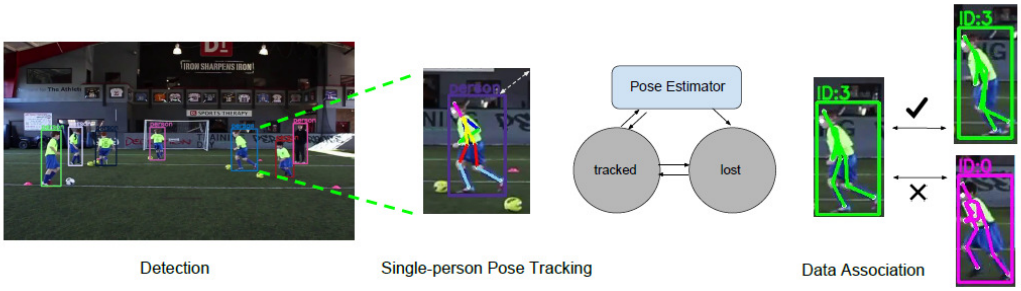


Figure 3.11: Visualisation of the lighttrack framework pipeline, source:[14]

OpenPose

The goal of this open-source pose estimator was to help the research community by providing an easy-to-use tool [5]. It supports multiple hardware configurations, different OS platforms, and many types of input/output options.

In addition to the generally used estimator model with 18 keypoints shown in 3.5, OpenPose also implements face, foot, and hand detectors. The default model used in

this algorithm consists of standard body keypoints plus foot variables, like heel and toe positions. Together, it creates 25-point format, demonstrated in 3.12.

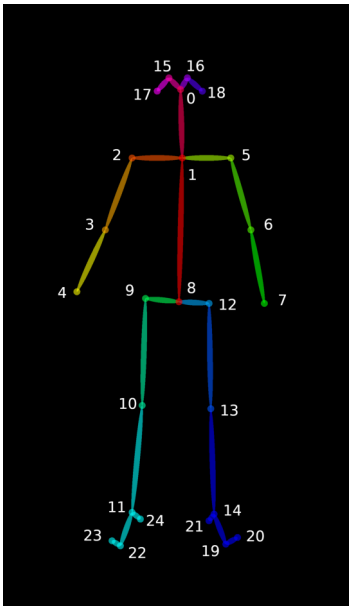


Figure 3.12: „Body 25“ model developed by OpenPose team, source:[5]

OpenPose utilizes the bottom-up approach to pose estimation with the theory of **Part Affinity Fields** . PAFs are 2D vectors that interpret the location and orientation of limbs. The whole body estimation is then put together from individual PAFs. This pipeline is shown in 3.13.

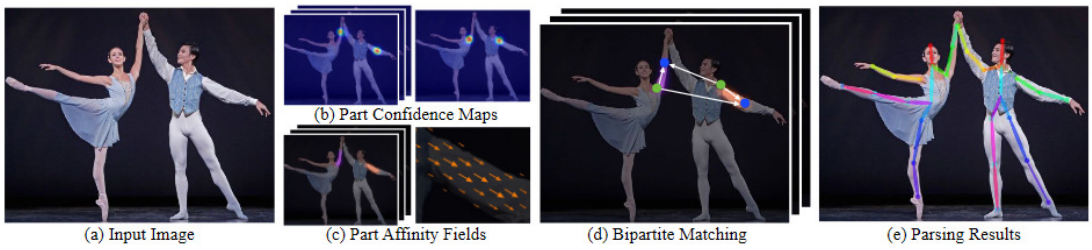


Figure 3.13: Visualisation of the OpenPose pipeline, source:[5]

What makes this framework so approachable is the diversity of available input methods. OpenPose works with single images, videos, live webcam stream, and even IP camera streaming. This means that the user does not have to implement his own pipeline for every use-case he encounters. Different modes are convenient to use with the ability to toggle them through command-line flags.

Similarly, multiple output methods are available, with the option to display them immediately or to save them to disk. The user can choose between the rendered video with the estimated pose, individual images corresponding to frames from the video, a heatmap output, or the most important one for researchers - JSON file structure.

Chapter 4

System proposition and implementation details

I aimed to create a generic enough application that would be freely available to anyone interested. It means that the system must work without any special hardware or software dependencies. Most people own a smartphone with a camera and a desktop computer, usually with Windows OS. Anything more would come with the cost of accessibility. Cross-platform code is not necessary, but welcome.

Most running gait analyses use two cameras, which is a bit of a complication. I have decided to attempt creating an application that would be able to work both with one camera only, but also two. This could be a nice compromise between ease of use and the additional functionality that comes with the second camera. The system will use primarily one camera capturing from the side of the runner and an optional one capturing from the side as used in [21], discussed in Section 2.3.

If those cameras capture the same portion of a run at the same time, the recording should be synchronized, which will provide additional information about the pose of the runner. It is important to keep the system fully functional without the need for synchronization and recording two views at the same time.

The final product should be a desktop application, where the user can upload a video of himself running, and his form will be analyzed. Problematic areas of his gait should be determined, visualized, and explained to him.

4.1 Choosing an algorithm

Choosing the correct pose estimator is critical for the accuracy of the proposed system. Luckily, for running gait analysis, real-time estimation and multiperson tracking are not necessary, both of which are still problematic in many cases. State-of-the-art algorithms nowadays have pretty similar mean precision for single-person pose estimation. Computational needs and efficiency are not a big deal either, because the algorithm will be used only once for each video and the outputs saved for further processing. Instead, I focused on accessibility and ease of use.

From the aforementioned algorithms and frameworks, OpenPose 3.2 stands out thanks to its additional foot keypoints, which could be important in running gait analysis because of the importance of foot position in this movement. It is also aimed at open-source developers, with a focus on portability and ease of integration into larger ecosystems. Furthermore,

it provides multiple output options. Data in easily parsed formats like JSON or XML are desired, as they can minimalize the amount of work required to connect the framework with the rest of the application. Image 4.1 and video formats are common amongst pose estimators, but still very much welcome. These can be later used in a graphical interface to visualize the work done by the algorithm.

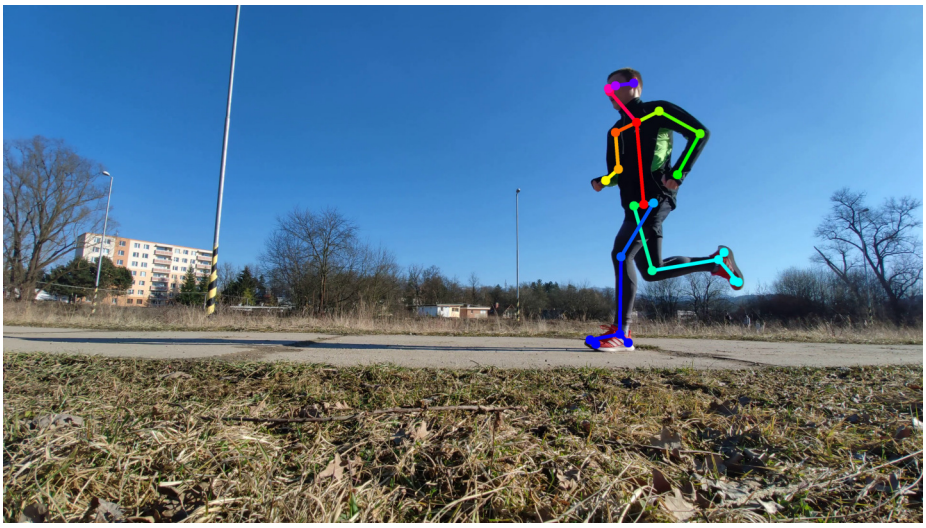


Figure 4.1: Image output example from OpenPose framework, using the „Body25“ model with additional foot and face key-points 3.12.

OpenPose generates a single JSON file for each frame of the input video. It has a „pose_keypoint_2d“ variable for each person detected in the frame, consisting of 75 floating-point values. These represent 3-tuples of **[x, y, confidence]**, one for each key-point. Their order is specified in the author's documentation[1].

```
"pose_keypoints_2d":[972.219,259.666,0.86884,
                     1101.44,347.985,0.826575,
                     1148.71,383.095,0.700951,
                     1113.37,542.192,0.250592,
                     1018.99,624.604,0.442225,
                     1048.69,312.554,0.823203,...]
```

Languages and tools

Python was chosen as the programming language for implementation, thanks to it being multiparadigm and having many libraries for working with large and complex data structures, advanced mathematical calculations, and even creating multiple types of user interfaces. All used additional packages were installed with python's package manager `pip`, and stored in a virtual environment[2].

[1] https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/02_output.md#keypoints-in-cpython

[2] https://docs.python-guide.org/dev/virtualenvs/

4.2 Video recording requirements

Even though the goal is to create a generic solution with no special requirements for the input video, some boundaries must be set. The camera recording from the side should be perpendicular to the direction the runner is going. The vertical position of the camera is not important, it can be placed directly on the ground, as long as the runner can be seen whole from head to toe. The recording device needs to be horizontally aligned with the ground, as that is the only way to determine the surface location. There should not be any movement of the camera, it should be static to eliminate undetectable changes in the coordinates system, which needs to be constant for the values to be comparable with each other.

The same rules apply for the posterior camera, with the exception that this one needs to be parallel to the runner's course or ideally lie exactly on the route.

There are no limitations in terms of resolution or framerate of the recording, but the landscape mode is better suited for this application. Standard 30 frames per second are fine, but the higher the rate, the better the analysis in the end.

Automatic view synchronization requires stricter rules to succeed. As demonstrated in 4.2 the posterior camera should be just on the edge of the field of view of the main camera or slightly beyond. The first few frames are used for this purpose. These recommendations should create an optimal environment for correct gait analysis, even without the synchronization option.

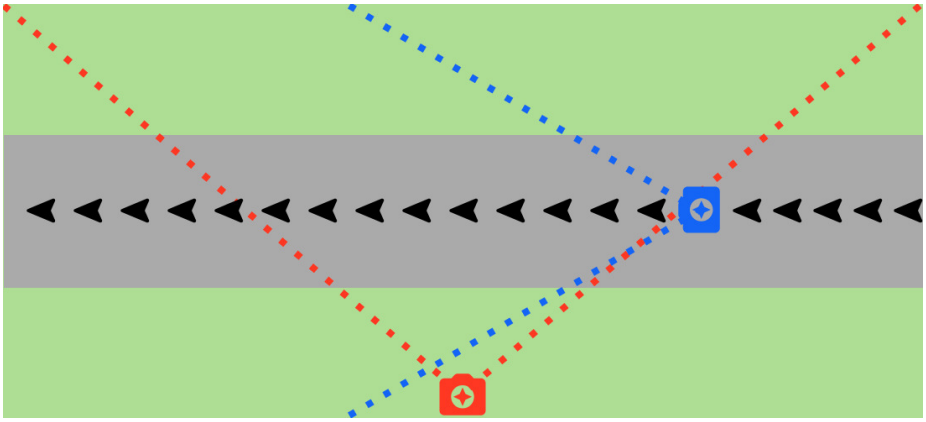


Figure 4.2: Expected positions of cameras for optimal recording and synchronization chance.

4.3 Mockup

No requirements for the user interface were specified, so I focused on simplicity and ease of use. The app is aimed mostly at newcomers and not as many long-term users, as running gait analysis should not be an everyday process. A mockup sketch of how the interface could look in the end is shown in 4.3.

It is crucial to allow seeing both views at the same time, as the synchronization function would not make sense without it. The user should be able to scroll through video frames one by one by himself, to see his form. After selecting a metric in the bottom-left corner menu, radio boxes appear, each representing one frame with an incorrect technique. After clicking one of them, the corresponding view should jump to the said frame, so it can be inspected

visually. The text box in the bottom-right corner will hold a few sentences explaining what the selected metric means and how it affects running.

Figure 4.3: Mockup of graphical interface layout for the final application

4.4 System modules

Before the start of the implementation, I drafted a rough idea of how the application would be split into modules. As can be seen in 4.4, the architecture should be similar to Model-View-Controller[3] design. This allows modifications to either part without affecting others too much. No communication should exist between the pose estimator, GUI, and back-end logic. Their connection should be handled exclusively by the controller module.

If desired, the used pose estimator could also be changed to another one. The only needed changes would be to the JSON loader module because other frameworks generate different formats.

Another possible future upgrade could also be changing to a different type of graphical interface, or even having multiple ones at the same time. It would enable users to choose their preferred form of the application. For example, switching between a web interface or a mobile phone standalone app. The neural network of the pose estimator is much more computationally heavy and takes a lot of runtime. Therefore, the option to upload not only the video to be estimated but also the saved output from the previous gait analysis would save a lot of time.

[3] https://en.wikipedia.org/wiki/Model-view-controller

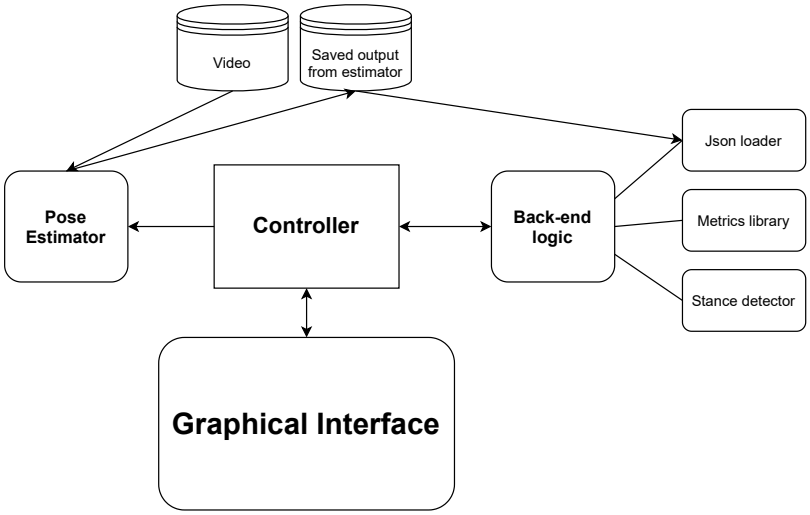


Figure 4.4: Architecture of proposed app modules and their connections

JSON loader

This module's input is a directory path with files containing JSON output from OpenPose estimator, explained in Section 4.1. Every file is parsed, cropped, and transformed into a more desirable format. A single frame array of keypoint 3-tuples is changed into a dictionary, where the key is a string with the name of the keypoint for better code readability and the value is a `Keypoint` class instance, which holds the tuple values as individual properties. These dictionaries are then merged into a list for easier passing amongst methods and simpler iterating through the whole structure. Empty frames with no person detected are filtered out at this point. A simplified version of the final data format is shown below.

```
data = [frame_1, frame_2, ..., frame_n]
frame = {
         "RKnee": [x,y,conf]
         "LWrist": [x,y,conf]
         ...
         "Neck": [x,y,conf]
        }
```

Pose estimator communication

OpenPose communication is implemented in the `estimator.py` file. The estimator's expected disk location is the root directory of this project. The path with video to-be-estimated is passed to the method. Its validity should be checked beforehand. OpenPose enables specifying input and output paths as command-line arguments. The estimator is called using the Subprocess[4] module of python's standard library. As the app is aimed at Windows users, the OS's Powershell[5] is utilized. Directory with output files is always saved to `outputs` folder located in the root directory. Its structure is demonstrated below.

[4] https://docs.python.org/3/library/subprocess.html
[5] https://docs.microsoft.com/en-us/powershell/

```
input name
├── video
├── json files
├── images
└── trajectories
```

Stance detector

This module is designed to detect side-view frames with the runner currently in a stance phase of the running gait. Stance happens when one foot is in full contact with the ground. For visual representation, check Figure 2.4.

Data structure created in JSON loader module discussed earlier serves as an input for the detector. Determining the presence of a stance uses two variables. The first one is the horizontal position of the foot. Contact with the ground cannot be established just from the pose estimator data, as the floor location is uncertain thanks to the changing nature of the camera position for each video. This is also the reason the cameras need to be horizontally aligned for the algorithm to work properly. Flat foot with small epsilon usually happens twice during running gait. Mid stance phase and during flight or „swing“ phase. Detected frames from a sample video with just this variable are shown in 4.5.

The second variable is used to filter out frames during flight and it is the rear leg tibia angle. As portraited in the figure, the rear leg is extended at the knee during flight, but in the stance phase it is bent to an approximately 90-degree angle. This value changes with each runner, so an epsilon equal to 25 is specified. The combination of these two variables generates frames with just stance, often a few in each phase. Module outputs a list of individual frame dictionaries.

The stance detector provides an additional option to merge frames into chunks. It adds another layer to the output structure, where concurrent frames are put into a standalone chunk, representing one stance phase. A list of these chunks is then returned.

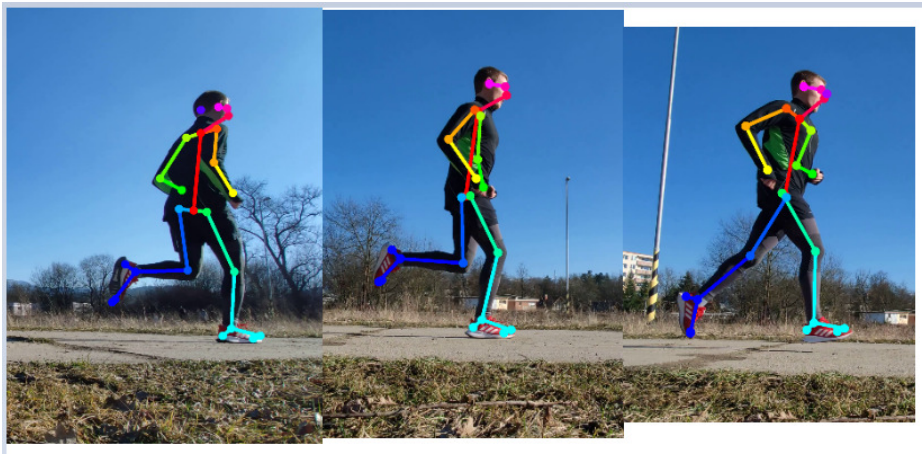


Figure 4.5: Horizontal foot detection from sample video, third frame is mid-flight and needs to be filtered out.

Metrics library

The library consists of multiple methods, each calculating single metric values for the whole data set. They all have the same signature, the parameters are data and boolean whether to return all values or just the irregular ones that need further attention. Data are in the standard structure created in the *JSON loader* module and used in the whole application. The majority of the metrics use data from the side-view camera, but some require the posterior recording. A list of dictionaries is always returned, where the key is the frame ID and the value is the metric calculated in the said frame.

Most metrics are determined by angles between specific keypoints, so assisting functions for angle calculations are implemented in `utils.py` source file. Method for determination of the runner's direction is also stored here.

Torso lean

Let us take a closer look at two of those metric methods. The first one is `torso_lean()`. As the name suggests, this method calculates the angle of the runner's torso. According to the gait analysis study mentioned earlier 2.3, the ideal torso lean for optimal running efficiency is about 8° but it may differ depending on the person's body structure. Boundaries for „good“ lean are therefore set from 2° to 10°, as at least a small forward lean is always desired. An additional value of 40° is used as a filter for pose estimator glitches to eliminate variables that should not exist because that amount of lean is almost impossible.

The angle of lean is calculated for each frame and if it does not land in the expected range, the frame is added to the dictionary returned from the function. Two keypoints in the dataset from the pose estimator are used to compute this value, specifically *Neck* and *MidHip*, portrayed in 4.6.

Pelvic drop

Pelvic drop is a metric that uses the posterior recording of a runner. It looks at the corresponding positions of pelvic bones on the horizontal axis. The pose estimator does not provide coordinates for these, but the closest relative keypoints are the hips. The horizontal angle between the left and the right hip is calculated for each frame and if the value is higher than the expected maximum, the frame is added to the dictionary and returned at the end, the same as in the torso lean function. This boundary is set to 6°. Hip key points are demonstrated in 4.7.

Trajectory plotting

The assignment specifies that there should be an option to plot trajectories for some joints that are important to running. This is done using the matplotlib[6] library in Python used to plot graphs and charts. Trajectory from the coordinates of a single keypoint is drawn as a line graph on a canvas that is then saved to a directory next to the output images, JSON files, and so forth.

[6] https://matplotlib.org/

Figure 4.6: Neck and MidHip key-points rendered on sample frame. The angle between them and the x-axis represents the lean of the runner's torso.

Figure 4.7: LHip and RHip key-points used for pelvic drop analysis from a posterior recording.

Controller

Implemented in `controller.py` file, the controller serves as the middleman between the graphical interface and the logic of the program. After the user uploads his video or saved data structure for the analysis and calls for it to be processed, the controller's `backend_setup()` method is invoked on the input path. How the logic of the method flows is demonstrated in 4.8.

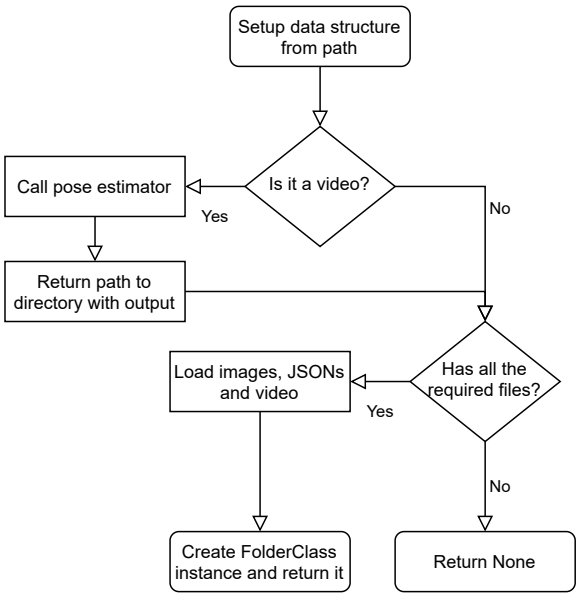


Figure 4.8: Flow chart for controller module.

Class `folderStruct` has multiple properties that hold:

- Path to video with the estimated pose
- Path to directory of images with the estimated pose
- Key-point data structure introduced in 4.4
- Path to directory with generated key-point trajectories (optional)

Its instance is used to pass all this data throughout the whole program.

The controller also implements a method that connects the automatic synchronization module with the interface. IDs of frames where the synchronization happens are returned in a dictionary, which the GUI uses to lock the views.

The last functionality stored here is the setup for the metric highlighting in the images in the graphical interface. When the user chooses a specific frame of a single metric, it is highlighted in the image to demonstrate visually its meaning. This method needs to determine which view to use, which frame, and which keypoints should be highlighted. Their exact coordinates depend on the resolution of the image because they are all converted into the same height and width. A list of *Keypoint class* objects is returned and later used to highlight the corresponding part of the frame.

View synchronization module

Automatic synchronization of the side and posterior view is highly experimental functionality and its precision depends on many factors. The main challenge here is the fact that the cameras do not have a constant position. Starting them at the same time would not work because they are at least a few meters from one another and a single person should be able to work with the system individually. Furthermore, every second of a video may take up to a few minutes to process by the pose estimator.

In professional filmmaking, multiple cameras are usually synchronized by sound. That is not applicable in this case, as the distance between the cameras might be too far.

The only option is to find matching poses from both views and align the corresponding frames. This raises the requirements for the camera position because the only output from it is a coordinate system that changes with every tilt of the device. A 90-degree angle between them is optimal, as is stated in Section 4.2.

The proposed synchronization pose is demonstrated in Figure 4.9. It is a point that can be identified from both views. When the runner is hitting the ground with one foot, the other leg rises upwards, making the angle of the knee of that leg smaller with each frame. The moment where it stops lowering is the frame of the synchronization.

From the posterior view, this point can be identified by the vertical position of the back foot ankle, compared to that same leg knee. While approaching this point, the runner's ankle rises on the **y** axis. When it stops, the synchronization point is found.

However, a single error by the estimator will break this method and there is no way to detect it happening. That is why there should be an additional option of manual synchronization by the user.

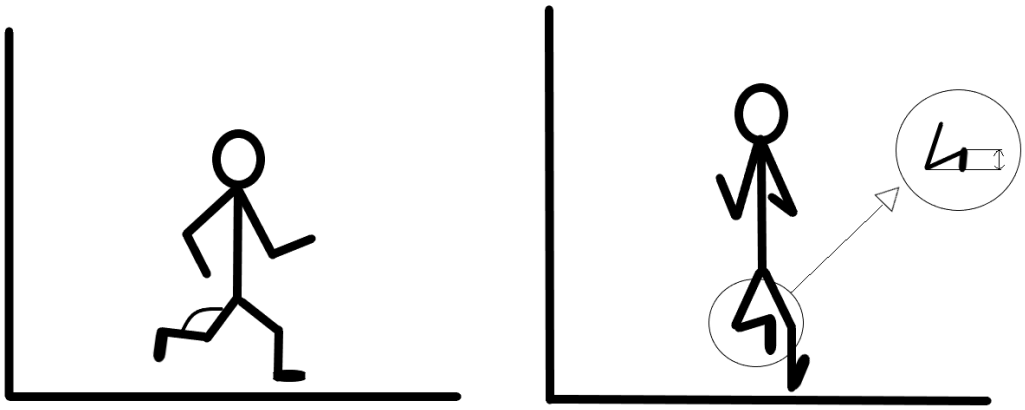


Figure 4.9: Synchronization point of both views.

Graphical user interface

The UI was created using the widely popular PyQt[7] cross-platform toolkit for graphical desktop interfaces. The final layout slightly differs from the initial mockup, as some functionality was added and some widgets redistributed for better usability. Python's `qdarkstyle` package is used to give the interface a simple and unified look. The default state after the start of the application can be seen in 4.10. All icons used in the interface are freely available at [8] and [9].

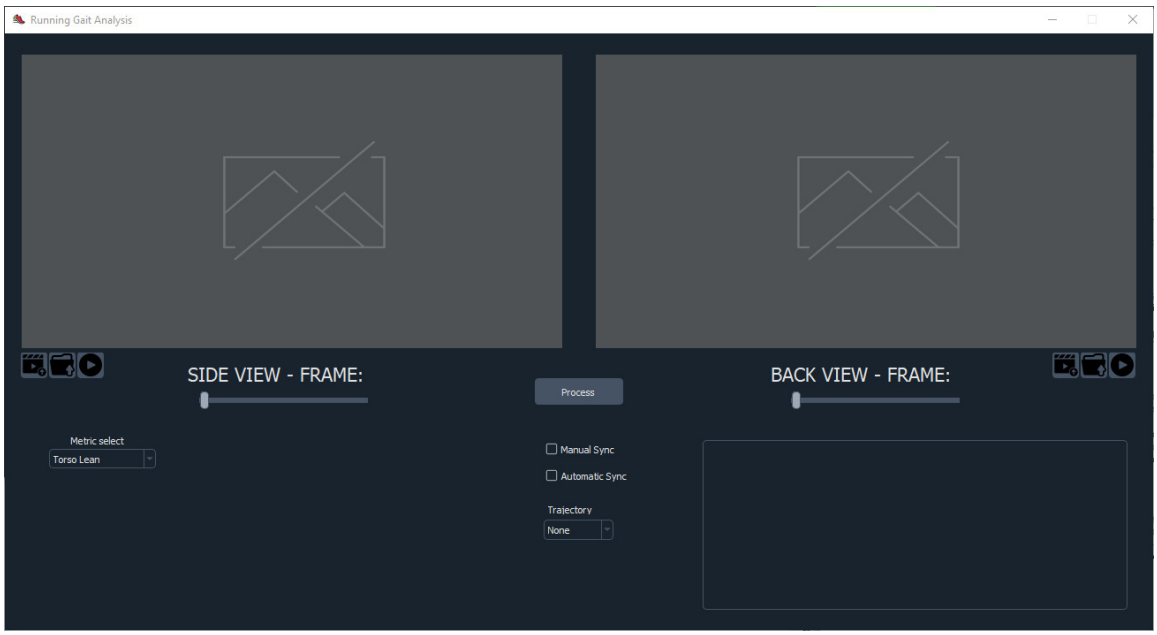


Figure 4.10: The graphical interface of the final application after the initial boot-up.

Firstly, the user needs to upload a video for either the side view or both of them. The two left-side buttons are file/folder pickers for a video or a directory with saved output from

[7] https://wiki.python.org/moin/PyQt
[8] https://loading.io/
[9] https://www.nicepng.com/

a previous analysis, respectively. After uploading, the process button calls the controller module to load all necessary data. If the estimator has to be called for a new video, this takes up to a few minutes.

Once the loading is done, the user can freely scroll through the frames of each video or press the third button underneath the frame that plays the whole video in slow motion. There are two synchronization options, which are only available if both views are active. The automatic sync uses the synchronization module that tries to determine a point where the frames align. If the button is checked, scrolling through one view also scrolls the other one to keep the views correctly aligned. This functionality is experimental and its success depends on the estimated video quality and the camera position. In the event of failure, there is an option to synchronize the views manually by scrolling both views to an alignment point and locking them with the *Manual sync* checkbox.

The joint trajectories created by the `trajectory.py` module can be visualized by choosing one from the combo box in the bottom-middle portion of the layout. They are drawn on top of the side view frame with a transparent background, as demonstrated in 4.11.

Figure 4.11: Trajectory of a left knee drawn on a sample frame.

Choosing a metric from the selection displays radio boxes with frames where the runner exhibits irregular values for the said metric 4.12. A closer description in a few sentences is written in the text box in the bottom right corner. It explains the meaning of the metric, its possible causes as well as its implications. For metrics calculated from the back view, if it is not available, a note is written to the user explaining he needs to upload a posterior video to enable these.

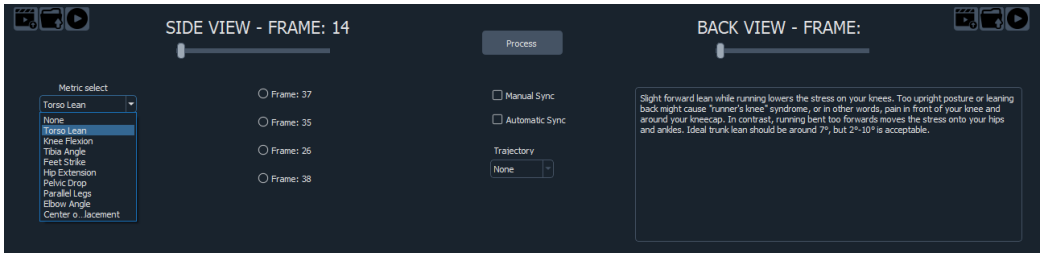


Figure 4.12: Metric selection and corresponding frames.

After selecting a single radio box, the corresponding view is moved to that frame. The parts of the body responsible for the current metric are highlighted in the picture (4.13) and the exact angle is written in the text box right beneath the description. This way the user sees everything needed for the analysis at the same time.

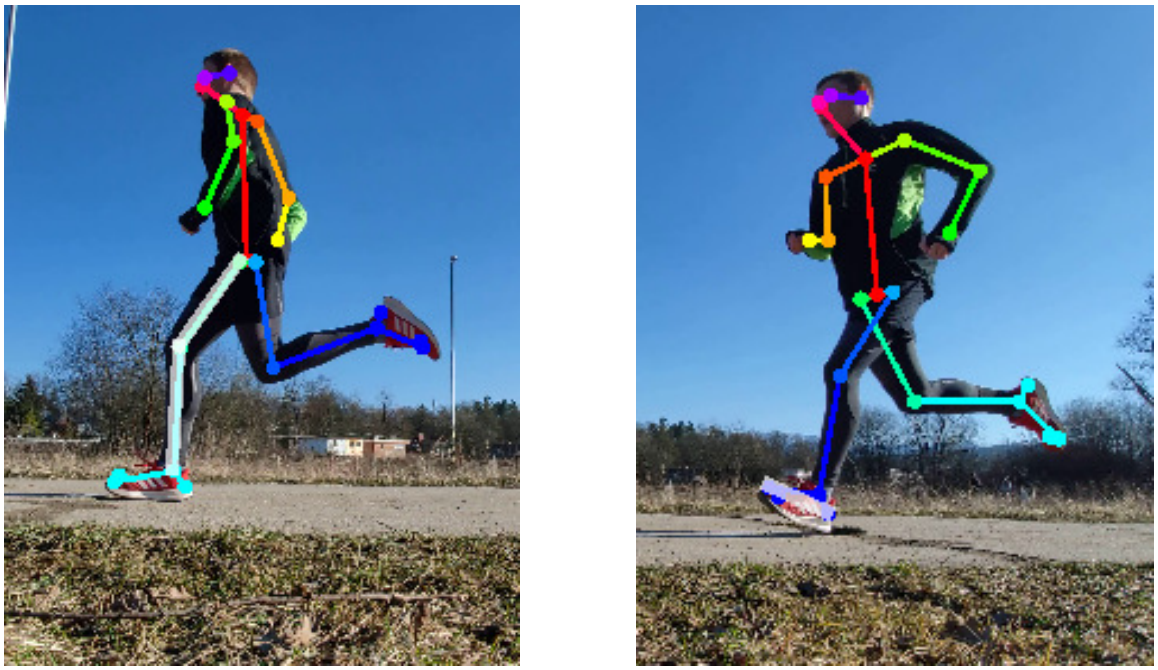


Figure 4.13: Examples of metric highlighting - knee flexion on the first figure and feet strikes in the second.

Chapter 5

Experiments and testing

The main goal of this paper was to create the back-end logic responsible for gait analysis. The user interface is a side product, therefore tests and experiments are aimed at this part of the product.

All videos used to develop and test the application were recorded using one of two mobile phones. The first one is an older Xiaomi Redmi Note 4[1] and the second one Xiaomi MI 9 SE[2]. Both are mid-range in terms of price, with a pretty generic camera that is not special in any way. Full HD resolution was used in most cases, with some exceptions that utilized a wide lens with a resolution of 3840x2160. Generally speaking, the rules introduced in Subsection 4.2 were abided and the side view camera was approximately two meters away from the runner to capture the whole body but in as much detail as possible.

5.1 Estimator accuracy test with body stickers

The paper of the chosen pose estimator OpenPose provides statistics stating that the mean average precision of keypoint estimation is very close to other bottom-up approaches. Nevertheless, this experiment was designed to evaluate the accuracy of a specific use case that is running gait.

Pink stickers were applied to the runner's body, specifically on his knee, ankle, and elbow. Three frames were picked randomly from this jogging sequence and their positions were extracted using a common photo editor. After applying the pose estimator algorithm, the coordinates of the specified joints were loaded from the JSON file of the corresponding frame. Comparison of the original image and the image with the rendered estimated pose can be seen in 5.1.

Coordinates of the actual joints and their estimated counterparts were compared and the results are demonstrated in 5.2. Measured accuracy turned out to be acceptable, as the failure rate of 0.5% should be neglectable. This will not affect the metric computations significantly.

[1] https://www.mi.com/in/note4/
[2] https://www.mi.com/global/mi-9-se

Figure 5.1: Three pairs of images like this were used to determine the accuracy of the estimator.

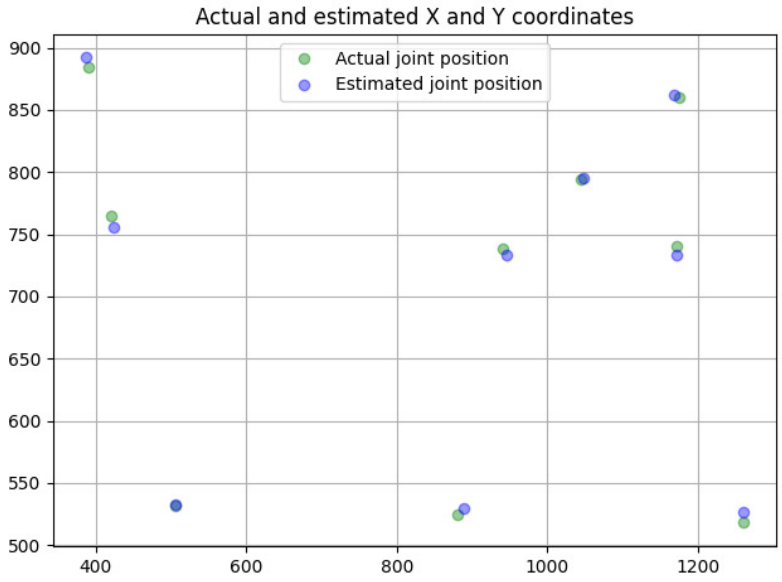


Figure 5.2: Results of the accuracy tests, for the pose estimator. In a standard full HD (1920x1080) recording all the tested joints were estimated with a miss rate of 0-10 pixels, which is less than 0.5%.

5.2 Stance detector tests

Many gait analysis metrics depend on the stance detector module. Its precision could be the deciding factor when evaluating the usability of the system.

Points of interest of this experiment are:

- Missed stance phases
- Incorrectly identified phases
- Number of phases per video
- Frames per single phase

Dataset for this experiment was constructed from 10 videos of the author running. All of them were recorded in 30 frames per second and full HD resolution except for two, which

were upgraded to 4k resolution. Some runs tried to imitate casual jogging pace and form, while others were purposefully wrong or different to try and catch edge cases of possible body position. For example, sprints with a long stride and body leaned forward or runs with kicking knees high into the air.

One special video was created with the intention to break the stance detector, thanks to the knowledge of its implementation details. The runner moved with a very little bend in the knees, essentially just jumping from one fully extended leg to the other.

The first experiment was aimed to determine how many stance phases are detected in one video. Because all data have similar proportions, this number should fluctuate around a single value. The results are shown in Figure 5.3.

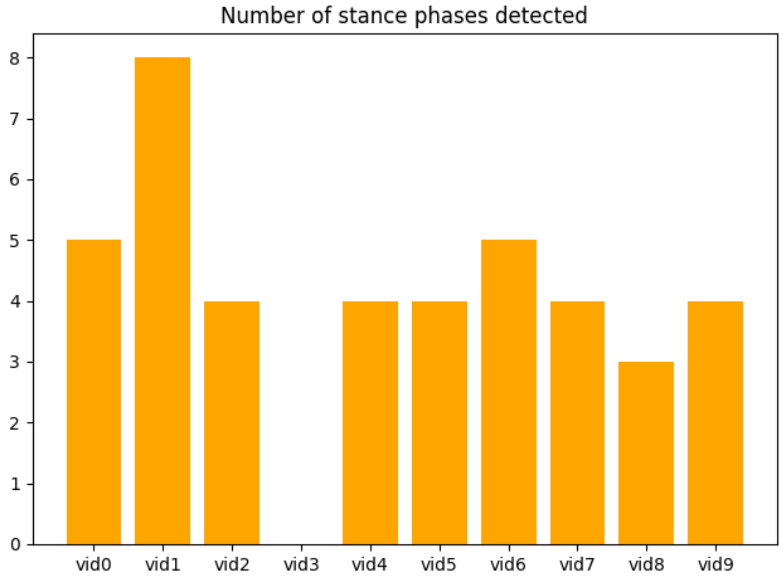


Figure 5.3: Number of stances detected per single video.

For most recordings, four stances were detected ± one stance. These small differences are often caused by the speed of the runner and can be neglected. However, videos with indices one and three stand out.

The third one with no stances detected was, as expected, the special recording with no knee bend. It proves that the feature can be broken with the knowledge of implementation details, but it should not be an issue, as no person runs like this naturally, and it would not make any sense to try improving his form regardless.

The next irregularity is a video with eight stances detected. We need to take a closer look at them to determine what happened. Their IDs are listed below.

```
[[40], [42, 43, 44], [52], [59], [61, 62, 63], [71], [78], [81]]
```

The problem lies in the functionality of merging stance phase frames into a chunk representing one phase. It happens only when frames are concurrent, therefore if a single frame is not recognized as a stance, it splits one phase into two.

Next, the number of frames per single chunk was calculated. This might indicate the trend of either missing stance phases or incorrectly detecting additional frames. All recordings had to be done in 30 frames per second because higher framerates would lead to more

frames per stance. As demonstrated in Figure 5.4, more than 40% of stance phases were detected as just a single frame. This could mean that redundant frames were also identified or that the splitting discussed earlier is happening often.

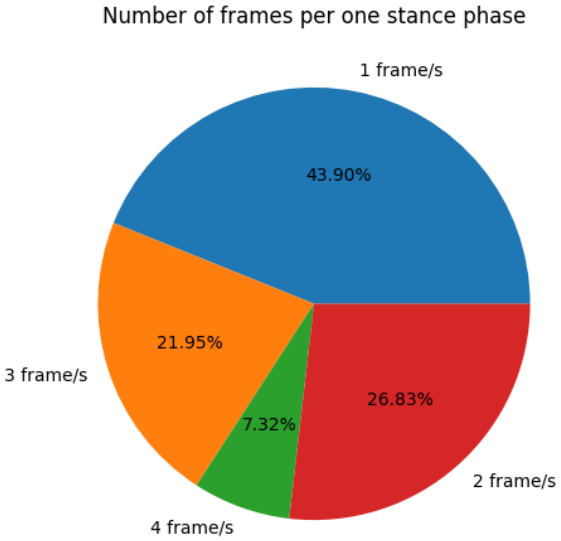


Figure 5.4: Number of frames per stance phase in 30fps videos.

To determine the cause, we should look at the distance between stances. Constant distances indicate correct detection, as the speed of the runner should not change dramatically during the small timeframe of the recording. The distance in frames is plotted in Figure 5.5.

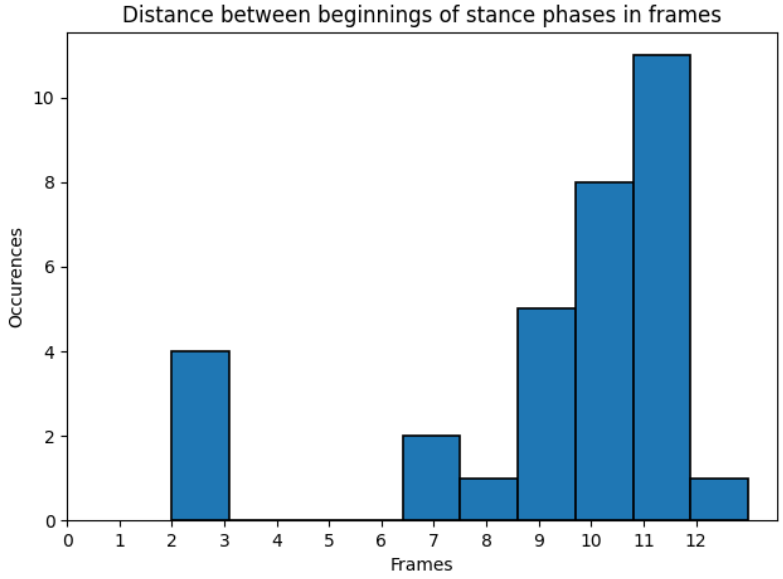


Figure 5.5: Distance between stances in frames.

The most frequent distances are in the range of 9 to 11. These will be taken as the baseline and other values observed one by one.

From the whole dataset, only two videos produced distances out of this scope. The first missed a single frame inbetween two others creating two stances from one. The other is a problematic video with highly irregular running form. It generated all other inconsistencies in phase distances. Pushing the knees as high as possible caused the runner to be in a pose similar to the stance phase while in the air, as can be seen in Figure 5.6.

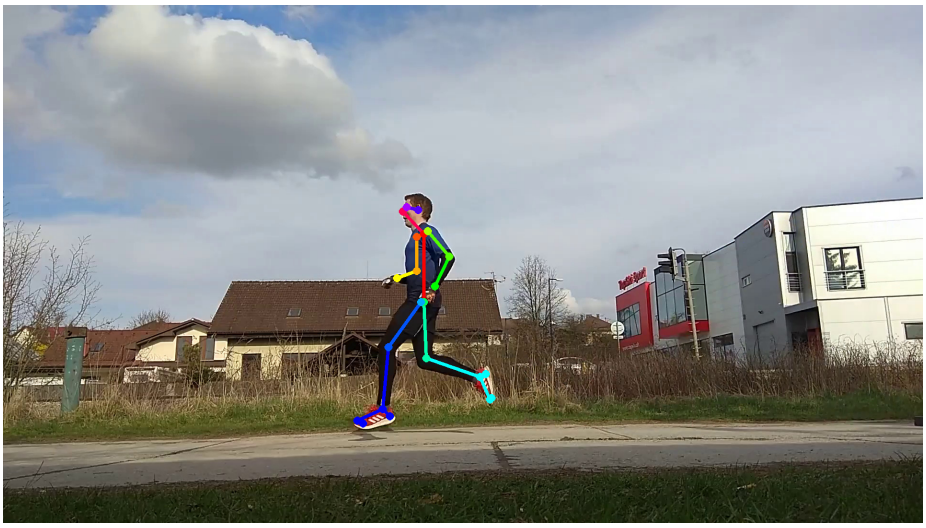


Figure 5.6: Irregular running form causing stance pose while in the air.

A stance phase with 3 identified frames is demonstrated in Figure 5.7.

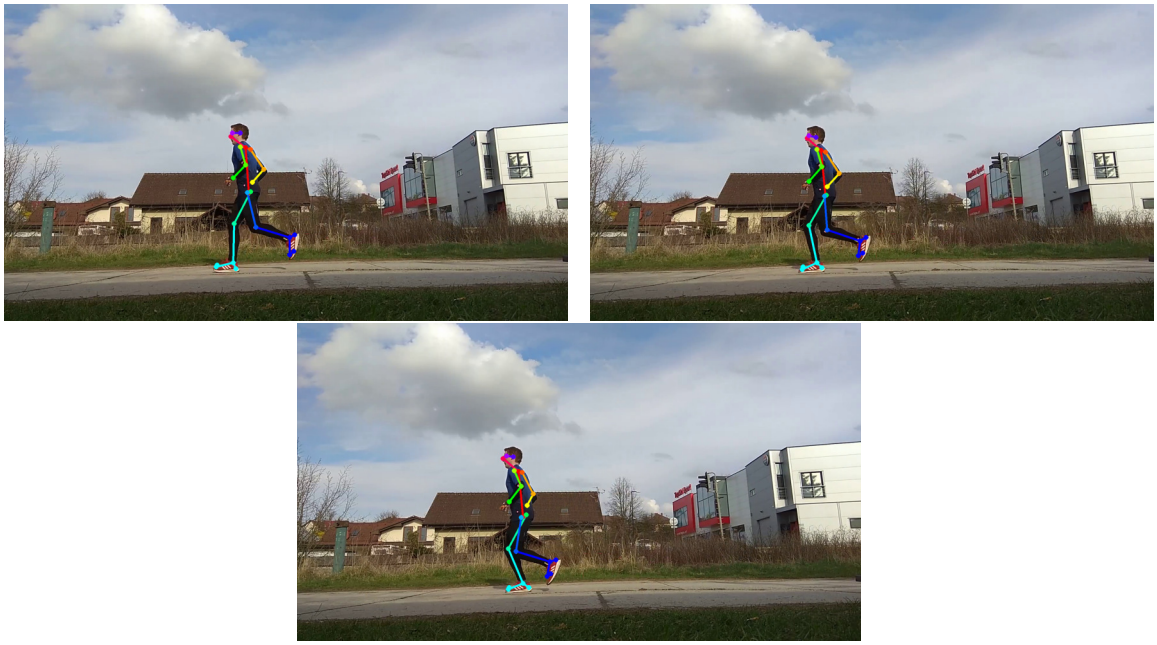


Figure 5.7: Three frames detected as a single stance phase of running gait.

5.3 Synchronization tests

To test the dependability of the synchronization module, two videos were imported into the system and attempted to synchronize. These were then visually observed to determine if it succeeded. In case of failure, the number of frames needed to forward one recording to achieve proper synchronization can be used to evaluate the accuracy of this module.

The determining factor turned out to be the accuracy of the pose estimator. When the runner starts appearing in the frame of the posterior view, the neural network has trouble assigning the correct joints to the detected keypoints. This occurrence can be seen in 5.8, where three concurrent frames are all detected very differently. A possible cause of this phenomenon could be the Bottom-up approach of the OpenPose algorithm, which tries to put together a human body from incomplete parts. A top-down framework might work better in this case, although it is just a theory and not a tested fact.

Figure 5.8: Irregular joint detection of a partial human body.

Let us take a closer look at some examples. Multiple sample pairs of videos were chosen from the author's dataset to evaluate this functionality.

In this particular one 5.9, the synchronization seems to work flawlessly, as the runner is in the same pose in both frames.

Although, as can be seen in the posterior view frame, the foot keypoints estimation is inaccurate. This proved to be a reoccurring trend in many tests. While it still worked out in this case, the ankle coordinates are used for detecting the synchronization point, as explained in Subsection 4.4. Therefore it might affect this module in a bad way.

Another successful synchronization happened on the two videos shown in Figure 5.10. The offset number of frames was determined to be equal to nine. The first two frames are the found synchronization point. The foot detection error discussed earlier is also present in the last pair of frames. It did not break the synchronization process, as it happened only afterward.

Figure 5.9: Automatic synchronization module success.

Figure 5.10: Successful synchronization shown in multiple moments from the recording.

However, the feature was not always accurate. In some experiments, it was a few frames off. For example, in the recording shown in Figure 5.11, the posterior view is three frames behind. This is exactly where the manual synchronization by the user comes in handy. All he needs to do is turn off the automatic synchronization, scroll the back view until they align, and turn on the manual synchronization.

Figure 5.11: Automatic synchronization module miss. The posterior view is three frames behind the side view.

5.4 Running motion analysis

The final experiment was conducted to evaluate the usefulness and possible applications of the developed system. Multiple videos from the aforementioned dataset were imported into the app and the runner's gait was assessed. A script in `tester.py` file was created to simplify the examination of the metrics values. It takes a directory with the JSON files from the pose estimator, loads, and evaluates them. All irregular angles are printed with their corresponding frame IDs. A simplified example can be seen in the listing below.

```
Stances:
[[26], [35, 36, 37, 38], [46, 47], [56, 57, 58]]

Torso lean:
{ 26: 13.302402825731718,
    38: 10.639906441994995}

Knee flexion:
{26: 17.490705289901, 47: 39.9847683951395, 58: 36.891791005889246}

Feet strike:
{34: 29.289844779065902}
```

An irregularity for every single metric was observed throughout the dataset at least a few times. With the knowledge of the metrics and their expected values, each of them was replicable purposefully. For example, torso lean exceeding the range of normal values, which is demonstrated in Figure 5.12. Another caught improper technique variables are shown in Figure 5.13.

The next goal was to examine runners with their generic gait and observe what the system suggests. To eliminate mistakes done in the fine-tuning process on the recordings of the author, videos from different runners were used at this point.

The first one is a seasoned long-distance runner going at his jogging pace. He exhibited no issues regarding extended tibia, foot strikes, or hip extension. With knee flexion, center of mass displacement, and elbow angle, only minor mistakes were detected. However, the main problem of this person is the lack of torso lean. Too upright posture was calculated during every stance phase. The exact values are shown in Listing 5.4. This could be caused

Figure 5.12: The lean of runner's torso below minimal value and above maximal value.

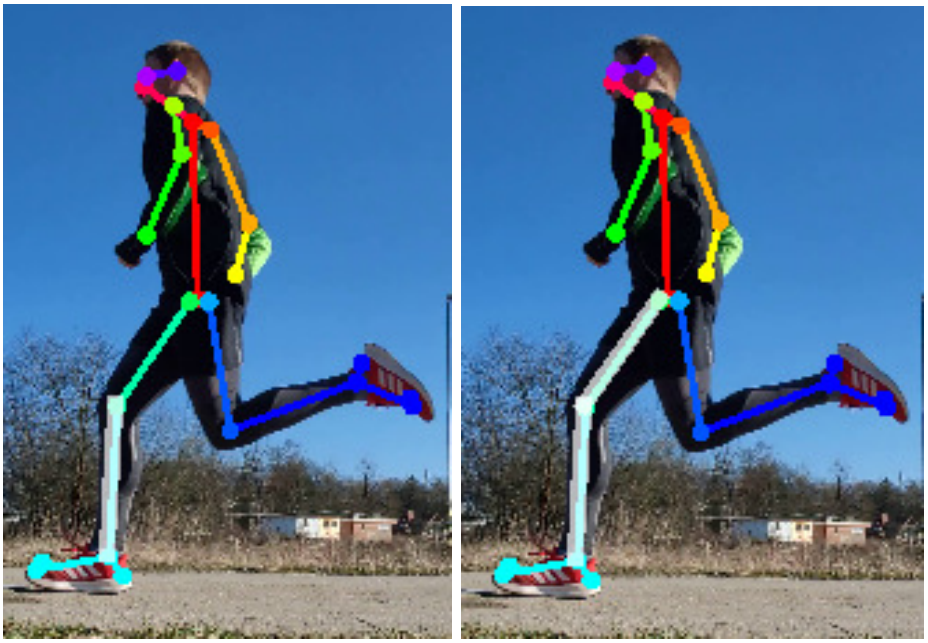


Figure 5.13: The first figure is showing a runner with a tibia angle at loading response equal to 93.07°, which is slightly too big and might cause excessive strain to joints. The second shows the knee flexed at 31.61°, exhibiting limited mobility.

by the nature of long-distance running and the goal to save energy. Although, he would still benefit from a slight tilt of his torso.

When the same runner was asked to sprint, his form changed vaguely. The trend of upright torso did not change. However, his tibia was slightly more extended, and the knees were a little less flexed during the stance phase. This happens due to the fact that

sprinting is focused solely on speed, which sometimes comes at a cost of proper technique. When jogging at a slower pace, the person has more time to focus on the correct form and movements. A single frame of this runner's gait is shown in Figure 5.14.

```
Stances:
[[28], [39, 40], [50, 51, 52], [61, 62, 63]]

Torso lean:
{ 28: -2.6765684808791974,
    39: -0.8122270865921308,
    40: 0.853022029446521,
    51: 1.6722523145649575,
    52: 0.8217264284727861,
    61: 0.023518305481360358}
```

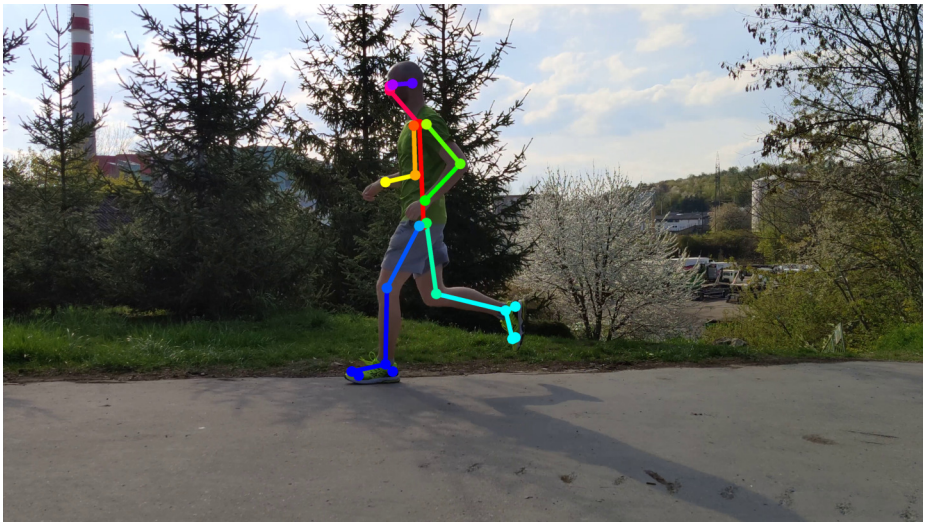


Figure 5.14: One frame during the stance phase. The runner exhibits correct elbow posture, but limited torso lean and overly extended tibia resulting in overstriding.

The second runner is fairly new to the sport and runs at a slower pace. Similarly, the main issue found in his technique is an upright posture with torso angles going as low as -2°. He also exhibited limited knee flexion a few times. Most other variables were in the norm, especially the elbow angle with not a single detected frame which is atypical amongst the performed tests. One frame from this runner's gait analysis is shown in Figure 5.15.

Posterior view

From the posterior view, the pelvic drop metric values stayed in the correct zone during most of the tests with a generic running form. However, when the runner purposefully leaned his body from side to side, the incorrect position was detected. An example is demonstrated in Figure 5.16.

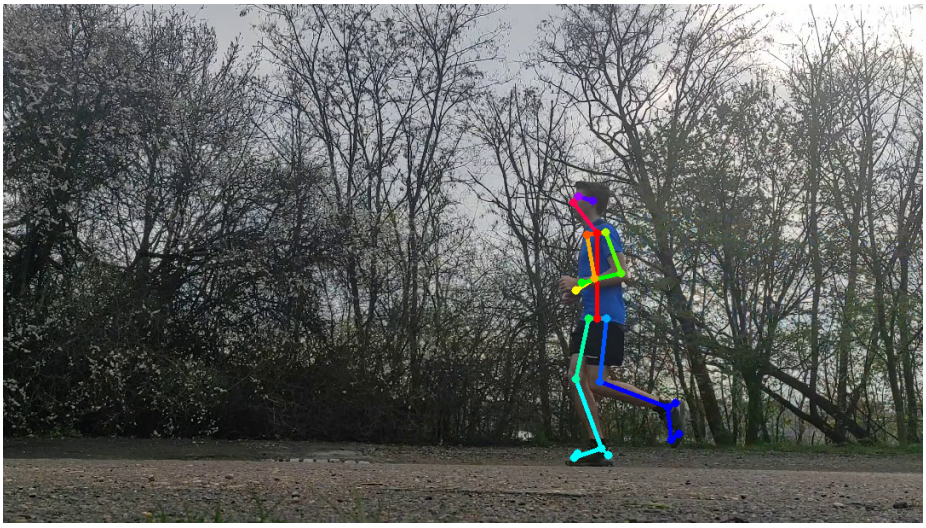


Figure 5.15: One frame during the stance phase. The runner has limited knee flexion at this moment, which might indicate excessive strain to the knees during running.

Figure 5.16: Pelvic drop equal to 7° that is over the expected values detected from the posterior view.

5.5 Verdict

The library of metrics calculations works as intended. Some minor inconsistencies come from the changing nature of the camera position and settings. It is also easily expandable in the future. An additional brief text evaluation of the form could be added in the future, so the user does not have to interpret the individual values on his own.

Metrics from the posterior view proved to be much harder to implement. This is caused by the static camera position because the person moves away very fast and there is little room to catch the exact body positions in good resolution. That can be solved by running on a treadmill, but that would eliminate the goal of the paper to make the system available

for a novice runner without special equipment. The other option is to move the camera dynamically, which would require another person and destroy any chance of the automatic synchronization process.

Chapter 6

Conclusion

The goal was to design a system that could be used to analyze running motion from two cameras. Firstly, the topic of human biomechanics and their application in running gait analysis in a clinical environment was discussed. Basics of neural networks and human pose estimation were studied afterward. Multiple algorithms tackling this area were explored and compared.

The final system was designed and implemented using the OpenPose framework. A library of methods calculating metrics used to evaluate the running form was created, which uses the output generated by the estimator. Multiplatform desktop application encapsulates this main logic in a nice and simple graphical user interface. Multiple experiments were conducted on a private dataset of videos with runners recorded on just mobile phones. The application is generic enough, so the user does not have to have any special equipment and can use it alone.

In the future, the metrics library can be easily expanded or altered. The design of the system enables the ability to change the graphical interface if desired. Alternative web or Android/IOS interfaces would give the user the choice of a preferred platform. The posterior view proved to be challenging for both the pose estimator and the metrics calculation as a static camera captures only a small part of the run accurately. The stance detector module could also be improved to determine every frame of the stance phase, not just a subset of them. The process of view synchronization is also problematic, but this functionality is not necessary for the rest of the application.

Bibliography

[1] Alderink, G. Joint Structure and Function: A Comprehensive Analysis. *Physical Therapy*. 4th ed. april 2006, vol. 86, no. 4, p. 598–599. DOI: 10.1093/ptj/86.4.598a. ISSN 0031-9023. Available at: https://doi.org/10.1093/ptj/86.4.598a.

[2] Alexander, R. M. *Mechanics of animal movement*. 2005 [cit. 2020-12-02]. Available at: https://doi.org/10.1016/j.cub.2005.08.016.

[3] Andriluka, M., Iqbal, U., Insafutdinov, E., Pishchulin, L., Milan, A. et al. *PoseTrack: A Benchmark for Human Pose Estimation and Tracking*. 2018.

[4] Babu, S. C. *A 2019 guide to Human Pose Estimation with Deep Learning* [online]. 2019 [cit. 2021-02-28]. Available at: https://nanonets.com/blog/human-pose-estimation-2d-guide/.

[5] Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S. and Sheikh, Y. A. *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. 2019.

[6] Cao, Z., Simon, T., Wei, S.-E. and Sheikh, Y. *Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. 2017.

[7] Clinic, M. S. . A. H. *Identify the cause and risk of running injuries: 3D running biomechanics* [online]. 2020 [cit. 2020-12-25]. Available at: https://www.msahc.com.au/news/february-22nd-2018-identify-the-cause-and-risk-of-running-injuries-3d-running-bi.

[8] CoCo. *Common objects in context* [online]. 2021 [cit. 2021-03-01]. Available at: https://cocodataset.org/#home.

[9] CrossFit. *Planes of Motion: Body* [online]. 2019 [cit. 2020-12-02]. Available at: https://www.crossfit.com/essentials/planes-of-the-body.

[10] Hall, S. *Basic Biomechanics*. 7th ed. McGraw-Hill Education, 2014. ISBN 978-0073522760.

[11] Jain, A. K., Jianchang Mao and Mohiuddin, K. M. Artificial neural networks: a tutorial. *Computer*. 1996, vol. 29, no. 3, p. 31–44. DOI: 10.1109/2.485891.

[12] Knudson, D. D. *Fundamentals of Biomechanics*. 2nd ed. Springer, 2007. ISBN 978-1-4419-6497-7.

[13] Mateo, A. *Expert Tips for Achieving Proper Running Form from Head to Toe* [online]. 2020 [cit. 2020-12-23]. Available at: https://www.runnersworld.com/beginner/a20811257/proper-running-form-0/.

[14] Ning, G. and Huang, H. *LightTrack: A Generic Framework for Online Top-Down Human Pose Tracking*. 2020.

[15] PapersWithCode. *Papers with Code* [online]. 2021 [cit. 2021-01-12]. Available at: https://paperswithcode.com/.

[16] Physiopedia. *Introduction to Human Biomechanics 1* [online]. 2020 [cit. 2020-12-02]. Available at: https://www.physio-pedia.com/index.php?title=Introduction_to_Human_Biomechanics_1&oldid=253499.

[17] Pipkin, A., Kotecki, K., Hetzel, S. and PhD, B. H. Reliability of a Qualitative Video Analysis for Running. *Journal of Orthopaedic & Sports Physical Therapy*. 2016, vol. 46, no. 7, p. 556 – 561. Available at: https://www.jospt.org/doi/10.2519/jospt.2016.6280.

[18] Saha, S. *A comprehensive guide to convolutional neural networks* [online]. 2018 [cit. 2021-02-11]. Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[19] Sharma, S. *Activation Functions in Neural Networks* [online]. 2017 [cit. 2021-01-10]. Available at: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[20] Sharma, S. *What the Hell is Perceptron* [online]. 2017 [cit. 2021-01-10]. Available at: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.

[21] Souza, R. B. An Evidence-Based Videotaped Running Biomechanics Analysis. *Physical Medicine and Rehabilitation Clinics of North America*. 2016, vol. 27, p. 217 – 236. DOI: https://doi.org/10.1016/j.pmr.2015.08.006. ISSN 1047-9651. Running Injuries. Available at: http://www.sciencedirect.com/science/article/pii/S1047965115000704.

[22] Stasiuk, A. *Pose Estimation. Metrics.* 2020. Available at: https://alexander-stasiuk.medium.com/pose-estimation-metrics-844c07ba0a78.

[23] Toshev, A. and Szegedy, C. DeepPose: Human Pose Estimation via Deep Neural Networks. *None*. June 2014, p. 1653–1660. DOI: 10.1109/CVPR.2014.214. ISSN 1063-6919.

[24] Tsang, S.-H. *Review: DeepPose — Cascade of CNN (Human Pose Estimation)*. 2019. Available at: https://towardsdatascience.com/review-deeppose-cascade-of-cnn-human-pose-estimation-cf3170103e36.

[25] West, A. *Yoga & Balance: Center of Gravity* [online]. 2017 [cit. 2020-12-23]. Available at: https://annwestyoga.com/yoga-balance-center-of-gravity/.

[26] WHO. *World Health Organization - Physical activity* [online]. 2018 [cit. 2020-11-04]. Available at: https://www.who.int/news-room/fact-sheets/detail/physical-activity.

[27] Wikipedia. *Computer Vision* [online]. 2021 [cit. 2020-12-28]. Available at: https://en.wikipedia.org/wiki/Computer_vision.

[28] YADAV, S. *Weight Initialization Techniques in Neural Networks* [online]. 2018 [cit. 2021-01-10]. Available at: https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78.

[29] YIN, L. *A Summary of Neural Network Layers* [online]. 2018 [cit. 2021-02-11]. Available at: https://medium.com/machine-learning-for-li/different-convolutional-layers-43dc146f4d0e.

Appendix A

Contents of the included storage media

The included storage media holds these files:

- **src/**: directory with all source files
- **examples/**: directory with example data for the application
- **requirements.txt**: List of Python packages required
- **setup.ps1**: Powershell script for the pose estimator setup

Appendix B

Manual

The project can also be downloaded from the public Github page[1].

B.1 Installation

First, you need to have Python3[2] with pip[3] installed. Then, the next steps are required:

1. Be in the root project directory
2. `python3.exe -m pip install -r requirements.txt` - this installs required python packages
3. `./setup.ps1` - this installs OpenPose estimator with its models

This setup process is created for the Windows OS. Additional options for Linux and macOS should be added in the future.

B.2 Usage

The application can be run from the root directory by - `python3.exe src/run.py`. An example video and already rendered output from the pose estimator can be found in the `examples/`. These can be used to demonstrate the functionality of the application.

[1] https://github.com/Radluy/Running-Gait-Analysis
[2] https://www.python.org/download/releases/3.0/
[3] https://pypi.org/project/pip/