BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

WEB APPLICATION FOR SETTING UP AN ERGONOMIC POSITION ON A BICYCLE

WEBOVÁ APLIKACE PRO NASTAVENÍ ERGONOMICKÉHO POSEDU NA JÍZDNÍM KOLE

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JIŘÍ VLASÁK**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. MARTIN ČADÍK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

154460

Institut:	Department of Computer Graphics and Multimedia (DCGM)
Student:	**Vlasák Jiří, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Computer Vision
Title:	**Web application for setting up an ergonomic position on a bicycle**
Category:	Computer vision
Academic year:	2023/24

Assignment:

1. Familiarize yourself with the issue of computer support for sitting ergonomics on a bicycle.
2. Research existing methods and applications for setting up the correct posture.
3. Design and implement a video and/or camera based web application for custom bicycle seating.
   Compare the approach based on the automatic detection of the rider's pose with the traditional marker-based motion capture method.
4. Experiment with the application, assess its properties in a user study, and/or further synthetic experiments. Discuss avenues for future development.
5. Present the achieved results in the form of a video, poster, article, etc.

Literature:

- Burt, P.: Bike Fit: Optimise Your Bike Position for High Performance and Injury Avoidance. A&C Black, 2014.
- Alberto Menache: Motion Capture Primer, In The Morgan Kaufmann Series in Computer Graphics, Understanding Motion Capture for Computer Animation (Second Edition), Morgan Kaufmann, 2011.
- Wade L, Needham L, McGuigan P, Bilzon J. Applications and limitations of current markerless motion capture methods for clinical gait biomechanics. PeerJ. 2022 Feb 25;10:e12995. doi: 10.7717/peerj.12995. PMID: 35237469; PMCID: PMC8884063.

Requirements for the semestral defence:
Points 1-3 of the assignment.

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

Supervisor:	**Čadík Martin, doc. Ing., Ph.D.**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	17.5.2024
Approval date:	9.11.2023

Abstract

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

Keywords

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

Reference

VLASÁK, Jiří. *Web application for setting up an ergonomic position on a bicycle.* Brno, 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Martin Čadík, Ph.D.

Web application for setting up an ergonomic position on a bicycle

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jiří Vlasák
May 1, 2024

Acknowledgements

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

Contents

Chapter 1

Introduction

Cycling is a massively popular activity around the world. People of all ages and skill levels enjoy it, from commuters to professional athletes. To enjoy cycling, it is important to have a bike that fits the rider. Having a bike that does not fit the rider can lead to unnecessary pain or over time even injuries. Having a bike that does not fit the can also have detrimental effect on performance. Therefore, a well fitted bike is important for professional athletes and amateurs alike.

Unfortunately, the setup of one's bike is not a trivial task. Information on how to setup the bike is often comprised of general guidelines and rules of thumb. These guidelines are often contradictory and can cause confusion. The setup of the bike is also highly individual and depends on the rider's flexibility, riding style, goals, etc. Similarly, each bike is different and requires different setup.

This led to the creation of a new profession called bikefitting. Bikefitters are experts who help cyclists set up their bikes. This was traditionally done in person and relied solely on the bikefitter's experience and rudimentary tools such as plumb bobs, goniometers, etc. However, in recent years, to make it more accessible, bikefitting has been increasingly relying on technology, mainly motion capture systems. This means bikefitters do not have to rely solely on their experience and can use data to make more informed decisions. However, these in-person bikefitting sessions are often not available in all areas and can be too costly for amateur cyclists.

To make bikefitting even more accessible, there has been a rise in bikefitting applications that aim to automate some of the bikefitting process. Despite the success of these applications, they are often limited to a single platform, provide only very basic information to the user, or are too costly.

This work aims to create a bikefitting application that does not suffer from these limitations. The application is available in the browser, provides the user with detailed information, while being easy to use and uses lightweight SOTA pose estimation models, that run directly in the browser to keep the cost of the application low.

Chapter 2 explains the measurements used in bikefitting and compares several commonly used software systems for bikefitting. Chapter 4 describes the pose estimation models used in this work. Chapter **??** evaluates the pose estimation models on videos of cyclists, filmed from the side, as this is the most common view used in bikefitting. Chapter 6 uses the best performing pose estimation model to create a new pose estimation model trained specifically for bikefitting. Chapter 7 describes the bikefit application and chapter 8 evaluates the application on real world data. Chapter 9 concludes the work and proposes future work.

Chapter 2

Bikefit

This chapter explains the standard process of bikefitting, the motivation behind it and compares several commonly used software systems for bikefitting.

2.1 An Introduction to Bikefitting

Cycling is massively popular activity world-wide. However, having incorrectly set position on the bike can lead to unnecessary pain and injuries. Having a position that does not suit the rider can also have a drastic effect on performance.

Due to these reasons, experts, known as bikefitters help cyclists with the setup of saddle position, handlebar position and sometimes even choosing the right parts such as saddle, handlebars or cranks.

Bikefit sessions are done mostly in person and while most professional cycling teams have a bikefitting specialist that helps to set the bikes for their athletes, they are often too costly for amateur cyclists.

This section draws from Phil Burt's Bike Fit 2nd Edition: Optimise Your Bike Position for High Performance and Injury Avoidance book [7].

2.2 Bikefit Measurements

This section describes the most common measurements used in bikefitting. These measurements can be accessed from a video using automatic bike fitting software and are simple enough to adjust by the rider themselves.

2.2.1 Saddle Height

The saddle height is a fundamental measurement that significantly impacts a rider's comfort and pedaling efficiency. It is argued to be the most important measurement in bikefitting and should be the first measurement to be adjusted [7]. Bad saddle height can even cause problems commonly associted with other measurements such as knee pain, lower back pain, neck pain, saddle sores, etc.

It is determined by considering the rider's knee angle at the bottom of the pedal stroke. Knee angle is the angle between the hip and the ankle measured at the knee joint. Burt recommends a knee angle of 35-40 degrees for average riders and even up to 30 degrees for professional cyclists [7]. Figure 2.1 shows the knee angle at the bottom of the pedal stroke.

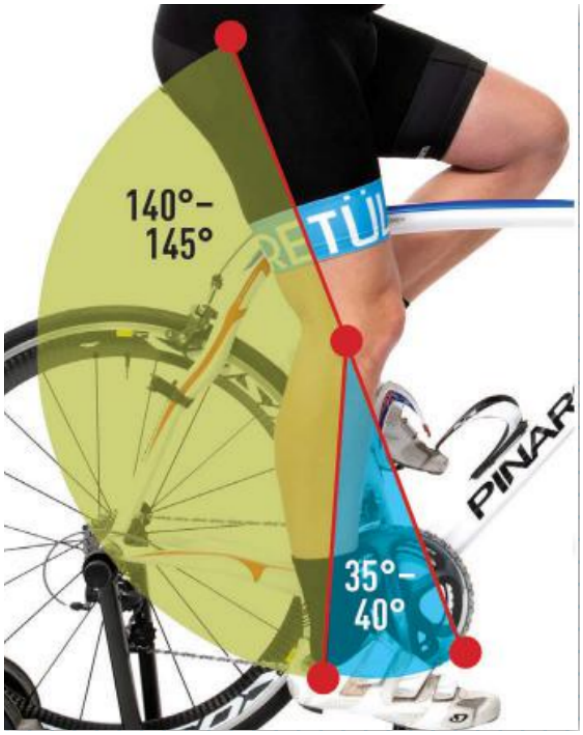


Figure 2.1: Knee extension angle of 140-145 degrees, which is often referred to as 35-40 degrees (being the angle of deviation from straight leg). Is optimal for the average rider. Taken from [7].

Higher saddle height can help to better recruitment of glutens and hamstrings, which can lead to more power output. However, it requires more flexibility and can lead to injury if the rider is not flexible enough. Similarly too low saddle height can increase the compressive forces on the knee and lead to pain and injury.

Proper saddle height is therefore a balance between power output and comfort. It is also important to note that the saddle height is not the only factor that affects the knee angle. The saddle fore and aft position and the cleat position also affect the knee angle.

2.2.2 Saddle Setback

Saddle setback or saddle fore and aft position refers to the horizontal position of the saddle with respect to the bottom bracket.

Setback is most often measured at the 3 o'clock position of the crank. At this position, the rider's knee should be directly above the pedal spindle. Having the knee too far back it is harder to generate power. Having the knee too far forward can lead to knee pain due to increased forces on the kneecap [7].

Saddle setback also affects the rider's balance on the bike. Having the saddle too far forward can lead to the rider's weight being too far forward, which can lead to hand pain because of too much weight on the handlebars. Having the saddle too far back can lead to the rider's weight being too far back, which can lead to make the front wheel feel light and make the bike harder to control.

Setback also affects the rider's hip angle. Hip angle is the angle between the shoulder and the knee measured at the hip. Having the saddle more forward can lead to a more open

hip angle, which can lead to more power output and more space between the rider's torso and legs at the top of the pedal stroke. This is why time trial bikes have a more forward saddle position.

2.2.3 Handlebar Height and Reach

Handlebar height measures how high the handlebars are in relation to the saddle. Handlebar reach measures how far the handlebars are in relation to the saddle.

Handlebar height influences mainly torso angle and to and also shoulder angle. Torso angle is the angle between the shoulder and the level plane measured at the hip. It is also known as the back angle. Shoulder angle is the angle between the hip and wrist measured at the shoulder. Handlebar reach influences mainly shoulder angle.

Handlebar height can be adjusted by changing the number of spacers under the stem or by changing the stem itself. Handlebar reach can be adjusted by changing the stem length or by changing the handlebars themselves.

While optimal handlebar height and reach are highly individual, there are some general guidelines. For example, a more upright position is more comfortable and is therefore recommended for longer rides. A more aggressive position is more aerodynamic and is therefore recommended for racing. Burt recommends back angle of about 45 degrees for average riders and shoulder angle of about 90 degrees with the elbow slightly bent [7]. For faster riders, the back angle can be lowered up to 30 degrees with more open shoulder angle. For more upright riders, the back angle can be increased up to 55 degrees with more closed shoulder angle.

2.3 Existing Solutions for Automated Bikefitting

2.3.1 MyVeloFit

MyVeloFit is a web application that uses pose estimation model to predict the joint locations for a side view video of the user pedaling their bike on an indoor trainer. Based on location of these joints, joint angles are then computed. On the basis of these angles and their relation to average angles, suggestions are made for adjusting the position of the saddle and handlebars.

The fitting process starts with the rider filling out questionaire about their mobility. This is then used to adjust the recommended angle ranges. For example: if the user has lower shoulder mobility, recommended ranges for shoulder angle will be increased so the user is not stretched forward so much.

After creating the user profile, user can create a fit session for one specific bike. In the process, the user selects their fit goal (performance, comfort, or balanced) and the type of bike they are using (road, gravel, mtb, triathlon, hybrid, or stationary). This also changes the recommended angle ranges.

Predicted keypoints

MyVeloFit predicts 6 joint locations for the camera facing side of the body. Most common pose estimation models predict similar keypoints. However, keypoints commonly used to adjust the position of the saddle, such as the heel and the fifth metatarsal of the foot, are missing. The keypoints used by MyVeloFit are: ankle, knee, hip, shoulder, elbow and wrist. Figure 2.2 shows the predicted keypoints.

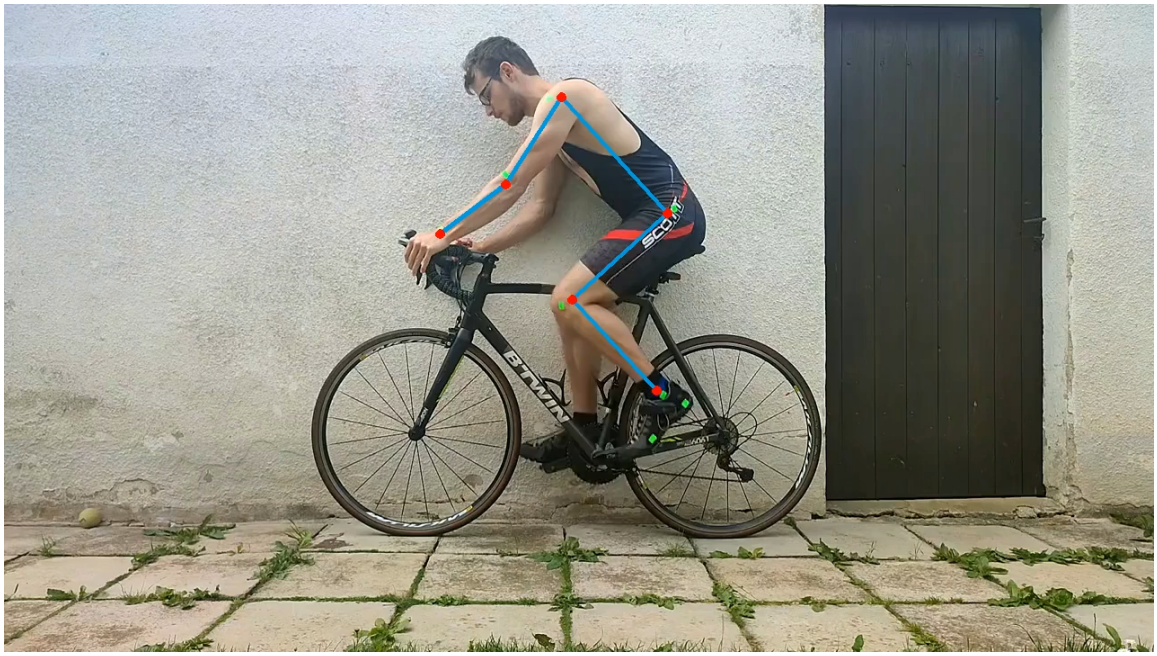


Figure 2.2: Side view image with predicted keypoints in MyVeloFit.

From the joint angles for every frame, some are selected for computing the joint angles at the top of the pedal stroke, front of the pedal stroke and bottom. Every position uses different angle ranges and even which angles are taken into account. Figure 2.3 shows the predicted joint angles at the top of the pedal stroke.

Based on the angles computed for parts of the pedal stroke, MyVeloFit then makes suggestions for saddle height, saddle fore and aft, handlebar height and handlebar reach. Figure 2.4 shows the recommended changes to the bike position.

Overall, MyVeloFit is relatively easy to use and its joint predictions are fairly accurate. However, it has few disadvantages:

- Only the most basic keypoints are used.
- Every video is converted to 30 FPS and cut down to 10 seconds.
- Video processing and keypoint predictions are slow (3-5 minutes).
- Requires subscription to get joint angles and recommended changes. Either a one time payment of 35 US dollars for access for 1 person and 1 bike for 2 weeks or 75 US dollars annually for unlimited number of bikes and people.

2.3.2 Retül

Retül is a bike fitting system employing 3D motion capture technologys. It utilizes infrared LED markers placed on specific body points to track the rider's movements dynamically while cycling. The led markers are tracked by multiple infrared cameras placed around the rider. The cameras surprisingly capture only 18 frames per second. Despite this research [30] shows that the system is relatively reliable compared to 3d motion capture systems with higher frame rates.

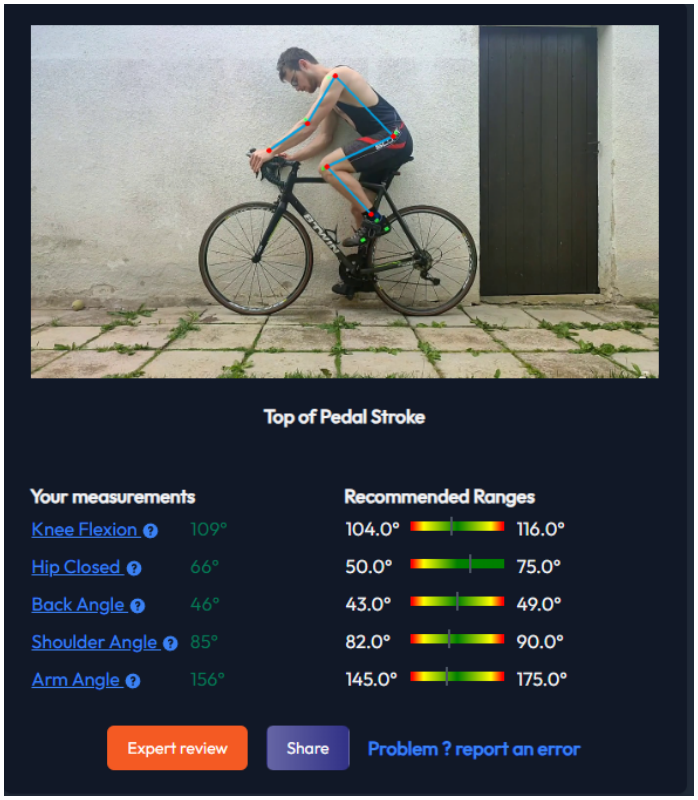


Figure 2.3: Predicted joint angles at the top of the pedal stroke in MyVeloFit.

Retül uses 8 markers placed on both sides of the rider's body. These markers are placed on the following locations: fifth metatarsal of the foot, heel, ankle, knee, hip, shoulder, elbow and wrist.

The markers are placed by the fitter on the rider's body. Accurate placement of the markers is crucial for the system to work properly. Even small deviations can lead to inaccurate results. Figure 2.5 shows the placement of the markers.

Retül's fitting process involves setting up the bike on a trainer equipped with the system. During the session, the rider performs various motions and pedal strokes while the Retül system captures real-time data on joint angles and movements.

Data Captured by Retül includes a wide range of joint angles and movements such as knee angles at top of the pedal stroke and bottom of the pedal stroke, hip angles throughout the pedal stroke, shoulder, elbow, and wrist positions in relation to handlebar reach and drop, as well as ankle and foot movement concerning cleat positioning and alignment. Figure 2.6 shows the captured data in Retül's software.

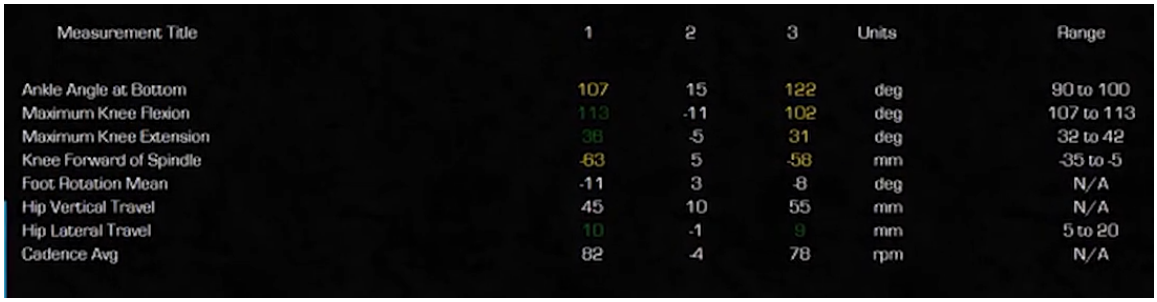
The normal ranges for these angles were constructed based on the data collected from thousands cyclists. However, these cyclists were not necessarily optimally fitted to their bikes. Therefore, the normal ranges may not be based on the optimal position for the rider.

Based on the captured data, Retül compares the rider's position to the normal ranges. Based on this comparison, the fitter can make changes to the bike position.

Despite the fact that Retül is a very popular bike fitting system, it has some important disadvantages:

Figure 2.4: Recommended changes to the bike position based on the angles computed by MyVeloFit.

Figure 2.5: Placement of the markers used by Retül. (Screenshot from instructional instructional video by Retül [29].)



Measurement Title	1	2	3	Units	Range
Ankle Angle at Bottom	107	15	122	deg	90 to 100
Maximum Knee Flexion	113	-11	102	deg	107 to 113
Maximum Knee Extension	36	-5	31	deg	32 to 42
Knee Forward of Spindle	-63	5	-58	mm	-35 to -5
Foot Rotation Mean	-11	3	-8	deg	N/A
Hip Vertical Travel	45	10	55	mm	N/A
Hip Lateral Travel	10	-1	9	mm	5 to 20
Cadence Avg	82	-4	78	rpm	N/A

Figure 2.6: Retül's software showing the captured data. (Screenshot from instructional video by Retül [28].)

- Costly equipment and setup requirements, limiting accessibility to some individuals or smaller bike shops.
- The need for trained Retül bike fitters to interpret and implement fitting recommendations effectively.
- Requires in-person fitting sessions. These sessions can be time-consuming and costly.

2.3.3 BikeFast Fit Elite

BikeFast Fit Elite is an iOS and Mac OS application that uses pose estimation model to predict the joint locations for a side view video of the user pedaling their bike on an indoor trainer. Compared to MyVeloFit, it uses additional keypoints for the fifth metatarsal of the foot and the heel. Figure 2.7 shows the predicted keypoints.

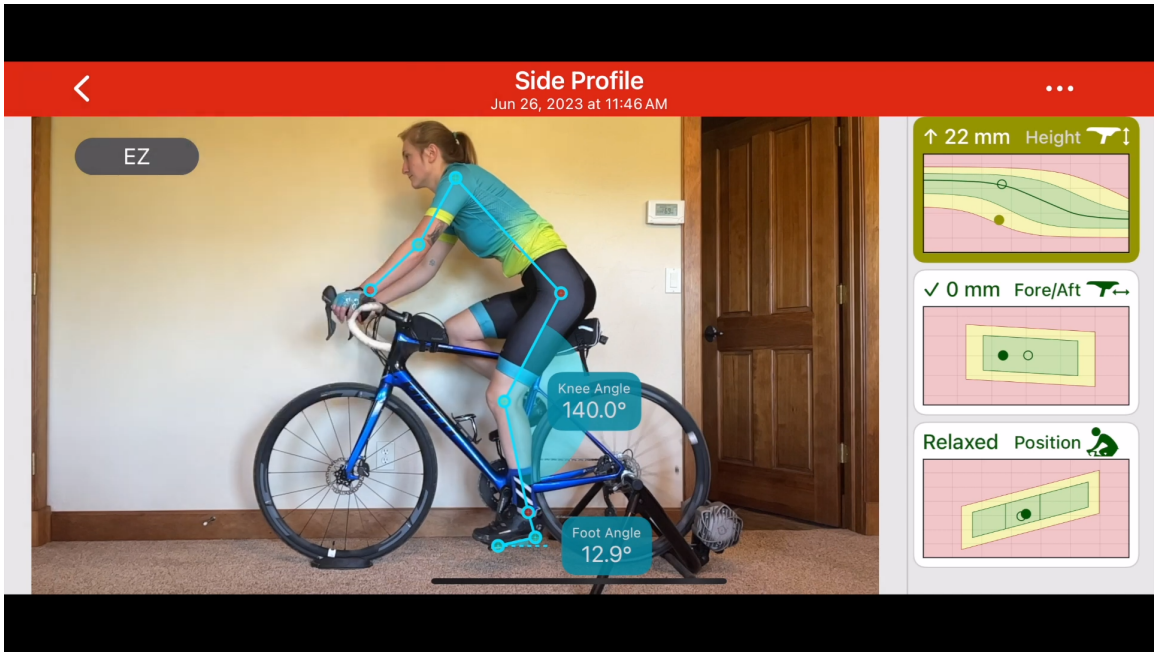


Figure 2.7: Side view image with predicted keypoints in BikeFast Fit Elite. (Screenshot from promotional video [5] by BikeFast Fit Elite.)

Similarly to MyVeloFit, it suggest changes to the saddle height and fore and aft position but it does not suggest changes to the handlebar position, arguing that the handlebar position is based on individual goals and flexibility.

Additionally, it also provides front view knee tracking to address possible knee wobble and asymmetry.

The app costs 19.99 US dollars and does not require a subscription. However, it is only available for iOS and Mac OS. Also it only captures 3.5 seconds of video.

2.3.4 Kinovea

2.3.5 Posiclist

Chapter 3

Marker-based Tracking

To estimate the keypoints, the application uses colored markers, which are placed on the parts of the body, whose position needs to be estimated (foot, heel, ankle, knee, hip, shoulder, elbow, wrist). The markers are at first localized manually by the user and their color is then used to detect them in the video. The markers are then tracked using a method similar to SORT. The detection and tracking is implemented using the OpenCV.js library [34].

3.1 Localization and Tracking Pipeline

3.1.1 Manual Localization of the Markers

At first, the user is asked to manually localize the markers by clicking on them in the first frame of the video. Figure 3.1 shows an example of the manual localization of the markers. The colors of these markers are saved and used for thresholding and detection of the markers in the following frames. The user is asked to first localize the marker on the foot, then the marker on the heel, then the marker on the ankle, etc. Figure 3.1 shows the manual localization process of the markers.

After clicking on a marker, multiple steps are performed:

1. The x,y coordinates of the marker are saved.
2. If there is already some marker localized and therefore some color thresholds defined and markers detected, the detected markers are searched if there is some marker close to the clicked marker. If there is, the location of the clicked marker is updated to the location of the detected marker. This is done so that user does not have to precisely click on the marker. This is especially useful when the markers are small and the user is using a mobile device with a small screen.
3. Based on the x,y coordinates of the marker, the color of the marker is extracted from the image.
4. The color of the marker is used to update the color thresholds used to detect the markers. More on this in section 3.1.2.

Figure 3.2 shows a closeup of the localization of the markers. The single localized foot marker is highlighted in green. The contours of the thresholded regions are highlighted in white. The centroids of the contours are highlighted by small green cross.

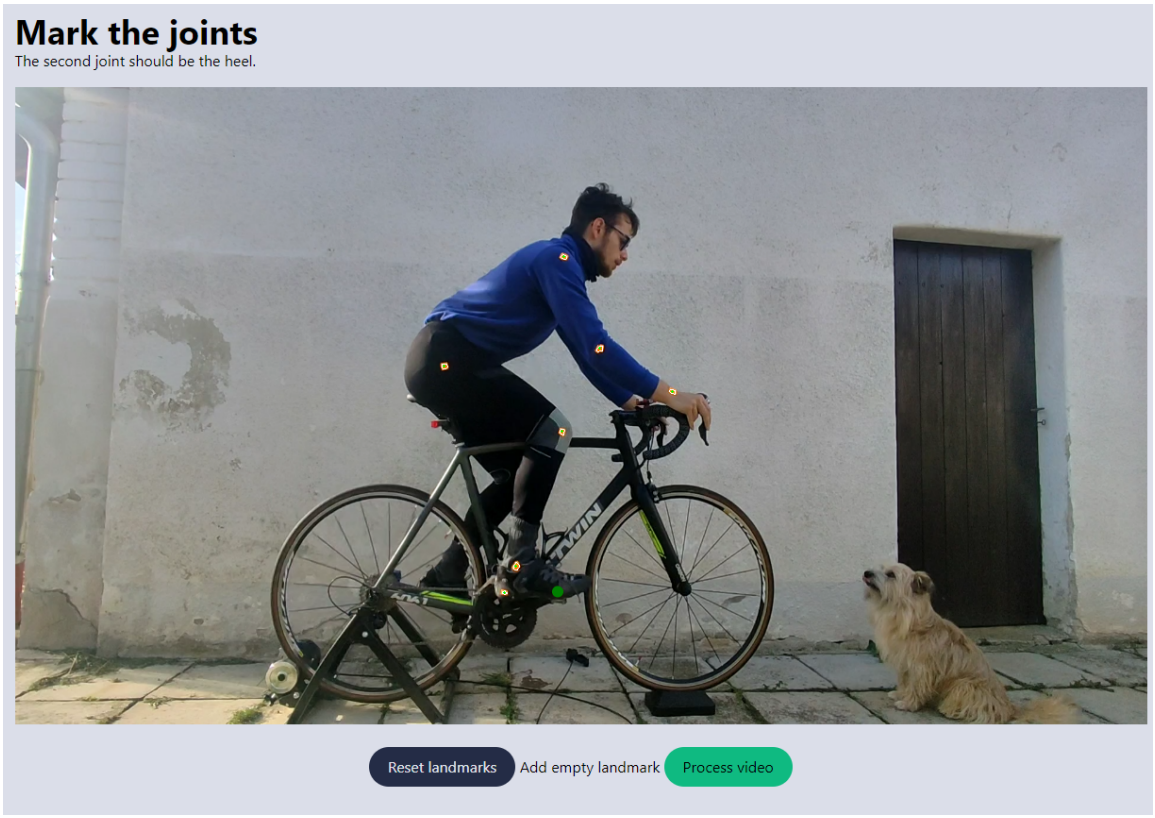


Figure 3.1: Example of the manual localization of the markers. The user is asked to click on the markers in the first frame of the video. The single localized foot marker is highlighted in green.

Figure 3.2: Closeup of the localization of the markers. The single localized foot marker is highlighted in green. The contours of the thresholded regions are highlighted in white. The centroids of the contours are highlighted by small green cross.

3.1.2 Detection

After obtaining the markers locations and colors, the markers are detected in the following frames. The detection is performed in the following steps:

1. The frame is converted to the CIELAB color space.
2. The frame is thresholded using the minimum and maximum A and B channel values of the localized markers and some margin. The margin is used to account for the changes in the lighting conditions. Empirically, the margin of 15 works well. Figure 3.3 shows an example of the thresholded image.
3. Contours are found in the thresholded image. The `cv.findContours()` method is used for this.
4. The centroids of the contours are calculated using the `cv.moments()` method.
5. The centroids are used as noisy detections.

The CIELAB color space is used because it is designed to approximate human vision. The A and B channels are used because they represent the color of the marker. The L channel is not used because it represents the lightness of the color and is not very useful for detecting the markers due to the changes in the lighting conditions. Experiments with using the H channel of the HSV color space were also performed, but there were some problems with the detection of the markers on the edges of the color spectrum (H=0 and H=255). The CIELAB color space does not have this problem, because it uses two channels for the color. Figure 3.3 shows an example of the thresholded image.

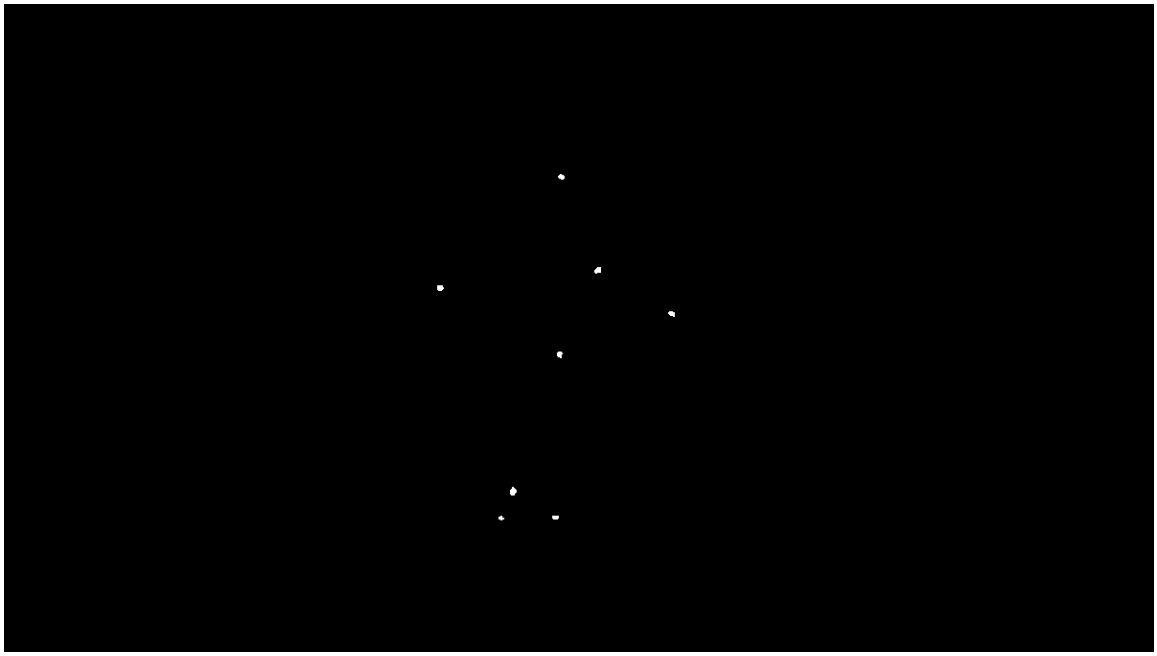


Figure 3.3: Example of the image thresholded using the minimum and maximum A and B channel values of the localized markers and a margin of 15. The markers are then detected as the centroids of the contours in the thresholded image.

3.1.3 Tracking

The detected markers are then tracked using a simple method similar to SORT [4]. The tracking algorithm works as follows:

1. Every possible assignment of the detected markers and the markers in the previous frame is created. To limit the number of possible assignments, the assignments are created only if the distance between the markers is less than 50 pixels.
2. For every found assignment, the cost of the assignment is calculated as the sum of the distances between the markers in the assignment.
3. To make tracking more robust, the cost of the assignment is increased by the difference of the angles between the markers in the assignment. This is done to prevent the markers from switching places.
4. Assignment with the most markers assigned and the lowest cost is selected.

Experiments with using the Kalman filter were also performed. Unfortunately I was not able to get the Kalman filter to work properly. The Kalman filter was not able to predict the position of the markers and the tracking was not very robust. The simple method described above works better.

3.2 Limitations

The marker-based approach was mainly chosen due to its theoretically perfect accuracy. However, in practice there are several limitations to this approach:

1. Accuracy of the system heavily depends on the placement of the markers. If the markers are not placed correctly, the results are not accurate. Since the placement is done by the user, this is a real issue. Correct placement of the markers is not trivial.
2. The user needs to have the markers available. They don't have to be special video markers, but they need to be of a different color than the background and the clothing of the user and need to be glued to the user.
3. The biggest limitation is the reduced robustness of the system. The system works well if the color of the markers is not present elsewhere in the image. However, in other cases, the markers are detected in the wrong places and the system does not work. Figure 3.4 shows an example of the failed localization of the markers. The markers are not localized correctly, because the leaves in the background have similar color to the markers.

Due to these serious limitations, the marker-based approach was not used in the final version of the application. And instead, the RTMPose pose estimation model is used. The details of the implementation are described in section 7.3.

Figure 3.4: Example of a failed localization of the markers. The leaves in the background have similar color to the markers and the markers are therefore not localized correctly. The markers are localized as the green dots.

Chapter 4

Pose Estimation Models

This chapter describes some of the good performing pose estimation algorithms and compares their performance on the task of side view pose estimation of cyclists.

Pose estimation is a computer vision task that involves predicting the locations of keypoints on a person in an image or a video. The keypoints are usually the joints of the person, such as the ankle, knee, hip, shoulder, elbow, wrist, etc. The pose estimation models are usually trained on datasets that contain images or videos with annotated keypoints. The models are then evaluated on the same datasets.

There are multiple approaches to pose estimation. The most common approaches are the top-down approach and the bottom-up approach. The top-down approach first detects people in the image and then predicts the keypoints for each person. The bottom-up approach first predicts the keypoints and then groups them into poses using hand writte post processing algorithms. Another approach is the one-stage approach, which predicts the keypoints directly from the image without the need for post processing. This work evaluates three pose estimation models using the top-down approach (RTMPose, HRNet and ViPNAS), described in sections 4.1, 4.2 and 4.4 and one pose estimation model using the one-stage approach (YOLOX-Pose), described in section 4.3.

Not all of the models predict the same keypoints. The models predict either 17 keypoints, as defined in the COCO dataset [22], 26 keypoints, as defined in the Halpe26 dataset [11], or 133 keypoints, as defined in the COCO-WholeBody dataset [18]. Figure 4.1 shows the sets of keypoints used by the MS-COCO dataset, the Halpe26 dataset and the COCO-WholeBody dataset.

4.1 RTMPose

RTMPose [17] is a pose estimation model from the authors of the MMPose framework [10]. It is designed to bridge the gap between the excellent performance of 2D pose estimation on public benchmarks and its application in the industrial community, which still suffers from heavy models and high latency.

The RTMPose models are designed to be lightweight and fast. The authors claim, that the RTMPose-m achieves 75.8% AP on COCO with 90+ FPS on an Intel i7-11700 CPU and 430+ FPS on an NVIDIA GTX 1660 Ti GPU. The RTMPose-s model, achieving 72.2% AP on COCO was also tested on a mobile device with the Snapdragon 865 chip, running at 70+ FPS.

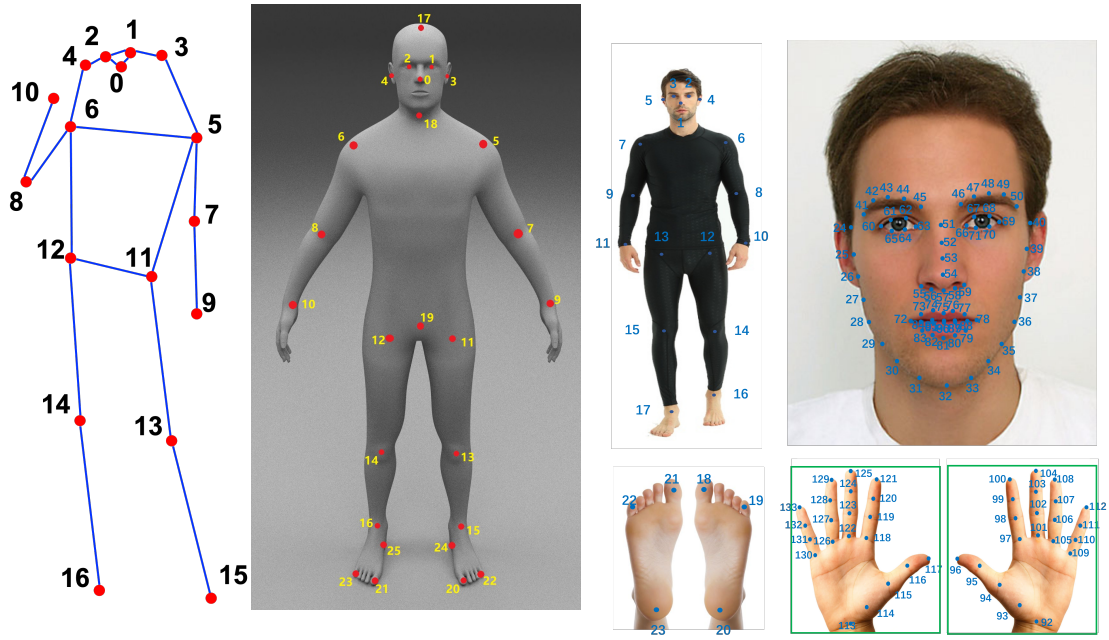


Figure 4.1: Sets of keypoints used by (from left to right) MS-COCO dataset, Halpe26 dataset and COCO-WholeBody dataset (images taken from [22] and [11]).

4.1.1 Architecture

The RTMPose models are based on the top-down pose estimation approach. They use a two-stage prediction, where the first stage is a person detector and the second stage is a pose estimator. The person detector is used to crop the image to the bounding box of the person. The pose estimator then predicts the keypoints for the cropped image. The authors claim that this approach is more accurate than the bottom-up approach, while still being faster in cases where the number of people in the image is lower than 6.

The CSPNeXt backbone is used in the RTMPose models. This backbone is primarly designed for object detection. Authors claim that backbones designed for image classification, are not optimal for dense tasks such as pose estimation, object detection, semantic segmentation, etc. Some backbones using high-resolution feature maps or advanced transformer architectures achieve good results, but suffer from high computional cost, high latency or difficulties in deployment. The CSPNeXt backbone is designed to have a good balance of speed and accuracy. Figure 4.2 shows the architecture of the RTMPose models.

For prediction of the keypoints, the RTMPose models utilize the SimCC [20] algorithm. SimCC reformulates human pose estimation as two classification tasks for horizontal and vertical coordinates. To reduce quantization error, SimCC can use a larger number of bins for the classification tasks. This can lead to better accuracy, while still being faster than post-processing methods commonly used with tradional heatmap based pose estimation models.

To better use spatial information, the RTMPose models use a Gated Attention Unit (GAU) [14] module. The GAU has faster speed, lower memory cost and better accuracy than the commonly used self-attention module, proposed in the Transformer architecture [37].

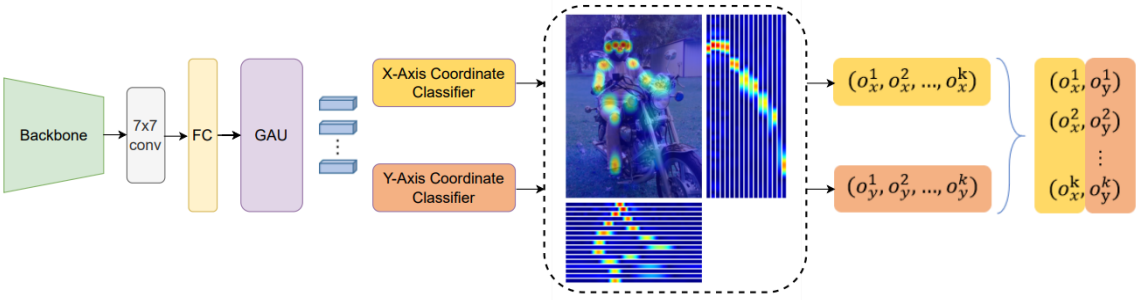


Figure 4.2: Architecture of the RTMPose models, which comprises of a CSPNeXt backbone, a convolutional layer, a fully-connected layer, and a Gated Attention Unit (GAU) designed to enhance K keypoint representations. Subsequently, the process of 2D pose estimation is treated as two separate classification tasks for the x-axis and y-axis coordinates. This involves predicting the horizontal and vertical positions of keypoints. Taken from [17].

4.1.2 Model Versions

This work evaluates the following RTMPose models:

RTMPose Body8 models: These models are trained and evaluated on the Body8 dataset consisting of 8 pose estimation datasets (AI Challenger [38], MS-COCO [22], CrowdPose [19], MPII [2], sub-JHMDB [16], Halpe [11], PoseTrack18 [1] and OCHuman [40]). They predict 17 keypoints, as defined in the COCO dataset.

RTMPose Halpe26 models: These models are also trained and evaluated on the Body8 dataset. However, they predict 26 keypoints, as defined in the Halpe26 dataset.

RTMPose WholeBody models: These models are trained and evaluated on the COCO-WholeBody [18] and UBody [21] datasets. They predict 133 keypoints, as defined in the COCO-WholeBody dataset.

4.2 HRNet

The HRNet (High-Resolution Network) [33] pose estimation architecture is characterized by its emphasis on maintaining high-resolution representations throughout the network. Unlike conventional methods that downsample the input image early in the network, HRNet adopts a multi-resolution approach, preserving detailed information essential for accurate pose estimation.

A notable feature of the HRNet pose estimation architecture is its ability to capture both local and global context effectively. The network consists of parallel branches, each processing different resolutions of the input image, and these branches are interconnected, facilitating the exchange of information between different scales. This design allows HRNet to address challenges associated with scale variations in human poses, ensuring robust performance in capturing both fine details and overall pose structure. Figure 4.3 shows the architecture of the HRNet models.

This work evaluates the HRNet-W32 model, which uses heatmap to predict the keypoints. The HRNet-W32 model is trained on the COCO dataset and predicts 17 keypoints, as defined in the COCO dataset.

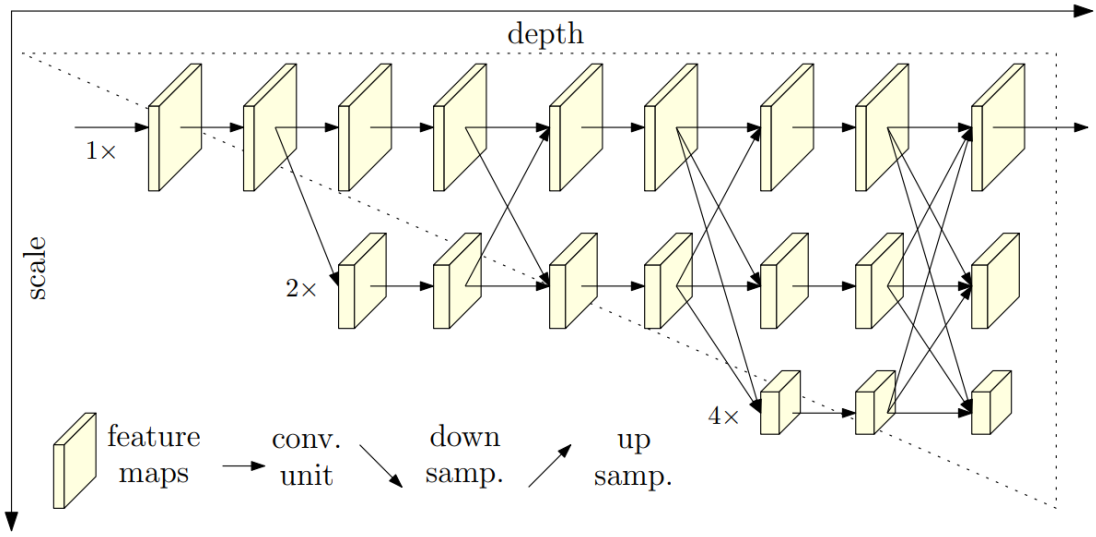


Figure 4.3: Architecture of the HRNet models. The HRNet models consists of three parallel branches, each processing different resolutions of the input image. These branches are interconnected, facilitating the exchange of information between different scales. Taken from [33].

4.3 YOLOX-Pose

YOLOX-Pose is a one-stage human pose estimation model that distinguishes itself from traditional top-down and bottom-up methods. The authors claim that YOLOX-Pose is the first one-stage human pose estimation model that achieves comparable performance to the state-of-the-art two-stage methods.

The one-stage approach has some advantages over bottom-up and top-down approaches. Bottom-up approaches first detect keypoints and then group them into poses. The grouping step is not learned and needs to be hand-crafted. This can lead to inaccurate results, especially in cases where the keypoints from different people are close together.

Top-down approaches first detect people and then predict keypoints for each person. This approach is favored by SOTA models, but the processing time grows linearly with the number of people in the image. This can lead to slow processing times in cases where there are many people in the image. SOTA methods also often use heavy detectors, which further increases the processing time.

The original YOLO-pose [27] model uses the anchor-based YOLOv5 detector [36]. The YOLOX-Pose model, which is evaluated in this work, uses the YOLOX detector [12]. The YOLOX detector is based on the YOLOv5 detector, but uses anchor-free approach, which does not require the anchor boxes to be predefined. This allows the YOLOX detector to be more flexible. The YOLOX detector generally outperforms the YOLOv5 detector on the COCO dataset. Figure 4.4 shows the architecture of the YOLOX-Pose model.

The YOLOX-Pose model is trained on the COCO dataset and predicts 17 keypoints, as defined in the COCO dataset.

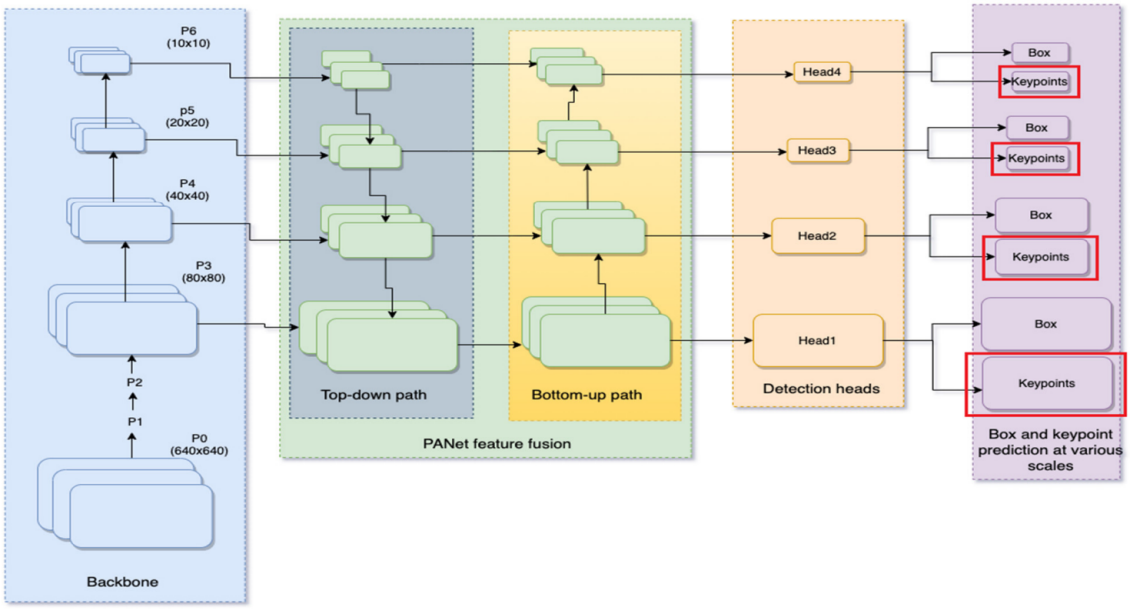


Figure 4.4: Architecture of the original YOLO-Pose model, which was based on the anchor-based YOLOv5 detector. The YOLOX-Pose model, which is evaluated in this work, is based on the anchorless YOLOX detector. Taken from [27].

4.4 ViPNAS

The ViPNAS (Video Pose Estimation via Neural Architecture Search) [39] model focuses on video pose estimation, employing two networks: S-ViPNet and T-ViPNet.

The S-ViPNet is used on invidual frames and keyframes in the video to predict the keypoints. The lightweight T-ViPNet network is used on non-keyframes to propagate the pose estimations. It extracts features from the current frame and also fuses heatmaps from the previous frame. The fused features are then used to predict the keypoints.

To find the optimal architectures for the S-ViPNet and T-ViPNet networks, the authors used neural architecture search (NAS). The search space for the S-ViPNet network is called the spatial-level search space. It consists of 5 dimensions (depth, width, kernel size, group and attention). The search space for the T-ViPNet network is called the temporal-level search space. It searches for the optimal fusion operation (addition, multiplication or concatenation) and the best stage of features to fuse.

This work evaluates the modified S-ViPNet model with the MobileNetV3 [13] backbone and SimCC [20] algorithm instead of predicting heatmaps. The model is trained on the COCO dataset and predicts 17 keypoints, as defined in the COCO dataset.

4.5 Results On The COCO Keypoint Dataset

To compare the potential of the methods, the results of the models on the COCO dataset are shown in table 4.1. All of the methods were trained only on the COCO dataset and predict the same keypoints. The results are taken from MMPose documentation [10].

The RTMPose models achieve strong results, with the RTMPose-l-256x192 model achieving the best results. The HRNet-W32 model also performs well but it is a much larger and

slower model. However, the other lightweight models such as YOLOX-Pose and ViPNAS-MobileNetV3 perform significantly worse than even the medium and small RTMPose models. Overall the RTMPose models show very promising results compared to the other models.

Table 4.2 compares the performance of the models with their computational costs. The RTMPose models are the most computationally efficient. Especially the m model, which achieves similaar results to the HRNet-W32 model, but is much faster and smaller. The YOLOX-Pose model computational costs are higher, because it is a one-stage model, which predicts the keypoints directly from the image. The ViPNAS-MobileNetV3 model is also very small and fast but it achieves significantly worse results than the RTMPose models.

Architecture	Input Size	AP	AP-50	AP-75	AR	AR-50
RTMPose-l	256x192	**0.758**	**0.906**	**0.826**	**0.806**	0.942
HRNet_w32	256x192	0.749	0.906	0.821	0.804	**0.945**
RTMPose-m	256x192	0.746	0.899	0.817	0.795	0.935
RTMPose-s	256x192	0.716	0.892	0.789	0.768	0.929
SimCC_ViPNAS-MBNetv3	256x192	0.695	0.883	0.772	0.755	0.927
Yoloxpose_m	640x640	0.695	0.899	0.766	0.733	0.926
RTMPose-t	256x192	0.682	0.883	0.759	0.736	0.920

Table 4.1: Results of the models on the COCO dataset. The results are sorted by the Average precision (AP). The AP describes the average precision over Object Keypoint Similarity (OKS) thresholds from 0.5 to 0.95 with a step size of 0.05. The AR describes the average recall over OKS thresholds from 0.5 to 0.95 with a step size of 0.05. The AP-50 and AR-50 are the same as AP and AR, but only for OKS threshold of 0.5. Similarly AR-75 is average recall with OKS threshold of 0.75. The results are taken from MMPose documentation [10].

Architecture	Input Size	AP	FLOPS(G)	Params(M)
RTMPose-l	256x192	**0.758**	4.2	28.1
HRNet_w32	256x192	0.749	7.7	28.5
RTMPose-m	256x192	0.746	1.9	13.9
RTMPose-s	256x192	0.716	0.7	5.7
SimCC_ViPNAS-MBNetv3	256x192	0.695	0.8	8.6
Yoloxpose_m	640x640	0.695	5.8	29.3
RTMPose-t	256x192	0.682	0.4	3.5

Table 4.2: Results of the models on the COCO dataset and their computational cost. The results are sorted by the Average precision (AP). The AP describes the average precision over Object Keypoint Similarity (OKS) thresholds from 0.5 to 0.95 with a step size of 0.05. The FLOPS describe the number of floating point operations needed to process the image. The numbers in the table are in billions. The Params describe the number of parameters in the model. The numbers in the table are in millions. The computational cost numbers are obtained using the MMPose library.

Chapter 5

Pose Estimation Dataset for Bikefitting

To evaluate and improve the performance of a pose estimation models in the setting of bikefitting, it is necessary to have a dataset that most closely resembles the real-world use case (videos of cyclists pedaling on an indoor trainer, usually in a side view). Since there is no such dataset, a new small dataset was created.

5.1 Video Acquisition

To construct the dataset, at first, the author shot 23 videos of himself pedaling on an indoor trainer or leaning on a wall and pedaling a bike. The videos feature both a road bike and a mountain bike. To ensure variety, they were shot on multiple days in different lighting conditions and enviroments, with various camera angles, clothes and riding positions. The videos were shot in FullHD resolution at 30 or 60 FPS using a Redmi Note 10 Pro smartphone. Both the standard and ultra-wide camera lens were used. The videos are 8-35 seconds long. Figure 5.1 shows examples of some of the videos present in the dataset.

5.2 Obtaining Ground Truth Annotations

Before shooting the videos, orange colored markers were placed on the camera-facing side of the rider's body. These markers were placed on the following locations: fifth metatarsal of the foot, heel, ankle, knee, hip, shoulder, elbow and wrist. Manually placing markers should be more accurate than annotating the keypoints in the video. The marker placement was guided by the instructions from the Retül bike fitting system [1]. These markers were the tracked using the methodology described in chapter 3. The application tracks the markers in a lower resolution. To get the ground truth annotations, the marker positions were then scaled to the original resolution of the video.

After obtaining annotations for the 8 markers facing the camera, the annotations for the remaining keypoints need to be obtained. To do this, the RTMPose-l-256x192 model was used to predict the keypoints for the videos. The model was trained on the Body8 dataset and predicts 26 keypoints. These keypoints are not needed for evaluation of the models for bikefitting, but are needed for the fine-tuning of the models. If the models

[1] https://youtu.be/kF0ZjtpO4FE?si=JRLX7lDNZ72oTaiC

Figure 5.1: Examples of some the videos present in the dataset. The videos feature both a road bike and a mountain bike. The videos were shot on multiple days in different lighting conditions and enviroments, with various camera angles, clothes and riding positions.

are fine-tuned only on the 8 keypoints, the model will unlearn how to localize the other keypoints. Another reason for predicting the keypoints is to get the bounding box of the person. The bounding box is needed to crop the frames to the bounding box of the person. The bounding box is predicted by the RTMDet-nano model [25] from the MMDetection framework [8]. Figure 5.2 shows an example frame from the dataset with the annotations.

5.3 Extracting Individual Frames

After obtaining the annotations, the individual frames are extracted from the videos. This is done since the pose estimation models are trained on individual frames and not on videos.

To reduce the size of the dataset and remove duplicate frames, not all of the frames are extracted. The frames to extract are selected using the k-means clustering algorithm [26]. Then only the centroids of the clusters are used as the frames for the dataset. The clustering is ran on the locations of the 8 markers. The number of clusters is set to 1/3 of the number of frames.

5.4 Removing Colored Markers From the Frames

To not bias the training and evaluation of the models, the colored markers are removed from the frames. This is done using an inpainting algorithm. The inpainting algorithm takes the frames with the colored markers and a mask of the colored markers and outputs the frames with the colored markers removed.

The mask of the colored markers is created the same way as in the marker-based tracking method, described in chapter 3. It works by converting the frames to the CIELAB color

Figure 5.2: Example frame with annotations from the dataset. Ground truth locations of the 8 camera facing keypoints are obtained by tracking colored markers. The remaining keypoints are predicted by the RTMPose-l-256x192 model. The bounding box of the person is obtained by the RTMDet-nano model.

space and thresholding the A and B channels using the minimum and maximum A and B channel values of the colored markers.

The inpainting was done using the opencv library [6]. The library contains implementations of the Navier-Stokes inpainting algorithm [3], which is based on fluid dynamics, and the Fast Marching Method [35], which works by propagating the information from the known parts of the image to the unknown parts. After testing both algorithms, the Navier-Stokes inpainting algorithm was chosen, as it produced better results. Figure 5.3 shows an example of the inpainting algorithm. It works well in most cases, but struggles with non-uniform backgrounds and can inpain unwanted parts of the image if the mask is not accurate.

5.5 Training and Validation Split

The extracted frames are split into training and validation sets. Frames from the same video are not split between the training and validation sets. The training set **1833** frames extracted from **19** videos and the validation set contains **476** frames extracted from **4** videos.

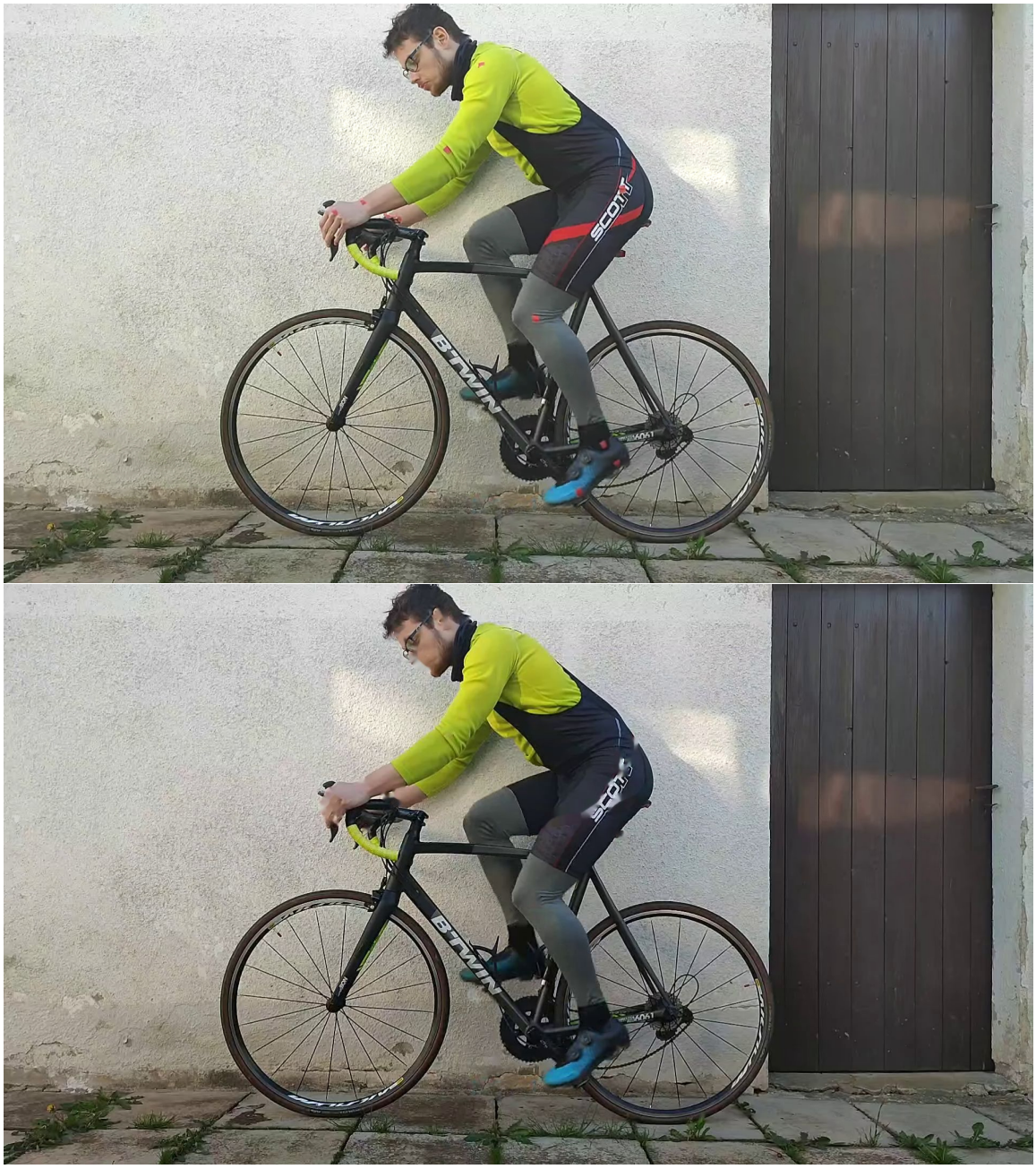


Figure 5.3: Example of the removal of the colored markers. The red colored markers are removed from the original image (top) using the Navier-Stokes inpainting algorithm. Most of the markers are removed well, but the algorithm produces blurry results on non-uniform backgrounds (pants with logo) and sometimes inpaints unwanted parts of the image (face), if the mask is not accurate.

Chapter 6

Fine-Tuning and Evaluation of Pose Estimation Models for Bikefitting

This chapter evaluates the performance of the pose estimation models on side view pose estimation of cyclists. Based on these results the best perfoming model is selected and fine-tuned on the bikefitting dataset. The fine-tuned model is then once again evaluated on the bikefitting dataset and compared with the pre-trained models. This model is then used in the bikefit application. Models are evaluated and fine-tuned on the dataset described in section 5.

The evaluation metrics are described in section 6.1. Initial evaluation of the standard, pre-trained models is done in section 6.2. The fine-tuning of the models is described in section 6.3. The results of the fine-tuned models are shown in section 6.4.

All of the evaluations and training is done using the MMDetection framework [8] and the MMPose framework [10].

6.1 Evaluation Metrics

This section describes commonly used metrics for evaluating pose estimation models and their limitations. The Normalized Mean Error (NME) metric, described in section 6.1.4, is used in the evaluation of the models on the bikefitting dataset.

6.1.1 Percentage of Correct Keypoints (PCK)

Percentage of Correct Keypoints (PCK) is a simple metric that calculates the percentage of keypoints that are within a certain distance from the ground truth keypoints. The distance is normalized by the size of the bounding box of the person. It provides a simple and intuitive measure of the accuracy of the model.

However, for the bikefitting dataset, commonly used thresholds of 0.1 or 0.05 are too big and the models achieved close to 100% PCK on the validation set. With a lower threshold, the models PCK would swing wildly if the distance between the keypoints and the ground truth keypoints is close to the threshold. This is why the PCK metric is not used in the evaluation of the models on the bikefitting dataset.

6.1.2 Object Keypoint Similarity (OKS)

The Object Keypoint Similarity (OKS) metric is a widely used metric for evaluating pose estimation models. It calculates the similarity between the predicted keypoints and the ground truth keypoints based on the distance between the keypoints and defined variance for each keypoint.

The equation for keypoint similarity is:

$$KS_i = \exp\left(\frac{-d_i^2}{2s^2\sigma_i^2}\right)$$

KS_i is the keypoint similarity for keypoint i

d_i is the Euclidean distance between the ground truth and predicted keypoint i

k_i is the variance for keypoint i

s is the scale of the object, s^2 is the area of the bounding box of the person

The final OKS is defined as sum of the keypoint similarities of visible keypoints divided by the number of visible keypoints.

It calculates a score between 0 and 1, where 1 means that the predicted keypoints are in the same position as the ground truth keypoints and 0 means that the predicted keypoints are far from the ground truth keypoints. However, it relies on defining the variance for each keypoint, which defines how much can each keypoint vary from the ground truth keypoint. Figure 6.1 shows the effect of the variance in the OKS metric for different keypoints. This setting of variance is dataset dependent and can be hard to define for a new dataset. This is why the OKS metric is not used in the evaluation of the models on the bikefitting dataset.

6.1.3 Mean Average Precision (mAP)

In assessing the performance of pose estimation models, the most widely used metric is the Mean Average Precision (mAP) which is calculated as the average of the precision-recall curve at different thresholds of Object Keypoint Similarity (OKS). This metric is often used in large scale datasets, such as COCO [22], where the keypoints of interest are not always visible and therefore are hard to accurately predict. Moreover, these datasets are labeled by multiple annotators, which can lead to inconsistencies in the annotations. This metric intuitively tells us how many keypoints are in some proximity to the ground truth keypoints. However, in the case of the bikefitting dataset, where only few well visible keypoints are of interest, the mAP metric may not be the most suitable for evaluation, since all of the keypoints should be in close proximity to the ground truth keypoints and we are more interested in the exact error of the keypoints.

6.1.4 Normalized Mean Error (NME)

As was discussed in the previous sections, the commonly used metrics for evaluating pose estimation models, such as PCK, OKS, and mAP, may not be suitable for the bikefitting dataset. This is why a simple Normalized Mean Error (NME) metric is used for evaluation of the models on the bikefitting dataset.

Normalized Mean Error (NME) is calculated by obtaining the Euclidean distance between the predicted keypoints and the ground truth keypoints, and then normalizing this

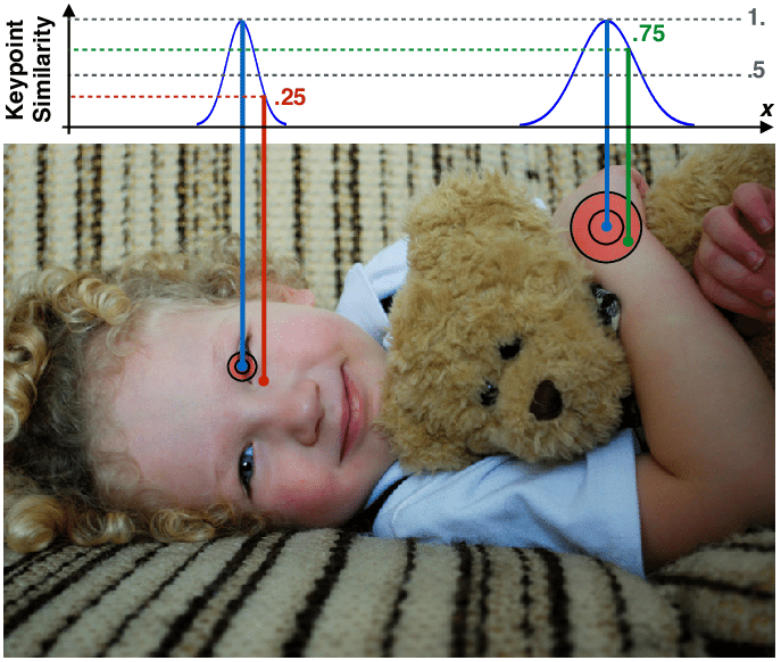


Figure 6.1: Diffent variances in the OKS metric for different keypoints. The keypoint similarity for the eye and wrist keypoints are different, even though the Euclidean distance is the same. Taken from [31].

distance with respect to the average bounding box dimension. The normalization is done to account for the scale of the video. It is then expressed as a percentage for better interpretability. The formula for NME is as follows:

$$\text{NME}_i(\%) = \frac{d_i}{\frac{s.w+s.h}{2}} \cdot 100$$

Where:

NME$_i$: is the Normalized Mean Error for the i-th keypoint, expressed as a percentage of the average bounding box dimension

d_i: is the Euclidean distance between the ground truth and predicted keypoint i

$s.w$: is the width of the bounding box of the person

$s.h$: is the height of the bounding box of the person

The NME for keypoint i is then averaged over all keypoints to obtain the NME for a single frame. The final NME is then calculated over all frames in the dataset. NME offers easy to interpret results, with a lower value indicating a more accurate prediction.

6.2 Initial Evaluation of Pre-Trained Models

This section evaluates the performance of the pre-trained pose estimation models on the bikefitting dataset. This is done, since there can be a significant difference in the per-

formance of the models on COCO dataset and the bikefitting dataset. The models are evaluated using the NME metric, described in section 6.1.

The models are evaluated using the MMPose framework [10], with the Inferencer API and the PyTorch backend. Topdown models (models that require inputs cropped to the bounding box of the person) use the RTMDet-nano model [25] from the MMDetetection framework [8] for person detection.

As was described in chapter 4, some models predict more keypoints than others. Most of the models are trained to predict 17 keypoints, as defined in the COCO dataset. The RTMPose, which are also trained on other datasets, inlcuding Halpe26, predict 26 keypoints. The WholeBody models predict 133 keypoints. The keypoints that are not needed for the evaluation are ignored. Since no models predict the fifth metatarsal of the foot, which is a crucial point for bikefitting, the small toe landmark is used instead.

6.2.1 6 Keypoint Evaluation

At first the models are evaluated using only the 6 keypoints that are both in the COCO dataset and the bikefitting dataset (camera facing ankle, knee, hip, shoulder, elbow and wrist). The results are shown in table 6.1. The best results are achieved by the RTMPose-m 384x288 model, but most of the bigger RTMPose models outperform the other models. This is in contrast to the results on the COCO dataset, where HRNet-W32 model performed similarly to the bigger RTMPose models. The relative decrease in performance of the HRNet-W32 model is probably caused by the fact that it was trained only on the COCO dataset, while the RTMPose models were trained on multiple datasets and are more robust to different poses and camera angles.

The best model achieves a mean NME of **2.12%**. This means that the average distance between the predicted keypoints and the ground truth keypoints is **2.12%** of the average bounding box dimension.

The good results of the RTMPose models are not surprising, since they were trained on multiple datasets, while the other models were trained only on the COCO dataset. The RTMPose models trained on the Body8 dataset perform slightly better than those trained on the Body8-Halpe26 dataset, which were trained to predict 26 keypoints. However, the models trained on COCO-WholeBody dataset perform significantly worse, this is probably because the WholeBody dataset is much smaller than the Body8 and standard COCO datasets.

The other lightweight models (YOLOX-Pose and ViPNAS-MobileNetV3) perform significantly worse than the RTMPose models. The YOLOX-Pose model is the worst performing model, with a mean NME of **2.74%**. The ViPNAS-MobileNetV3 model performs slightly better, but still worse than even the tiny RTMPose model.

Overall, we can see few unsurprising results. Models that are larger, or trained on multiple datasets, perform better than the smaller models trained only on the COCO dataset. It is interesting though, that there is not a huge difference between the large and medium RTMPose models, but there is a clear advantage to RTMPose models with larger input size. This is probably because the larger input size allows the model to capture more details in the image.

6.2.2 8 Keypoint Evaluation

The models which also output heel and foot fingers keypoints are evaluated on all of the 8 keypoints present in the bikefitting dataset. Meaning the camera facing fifth foot metatarsal

Architecture	Input Size	Trained On	Keypoints	NME (%)
RTMPose-m	384x288	Body8	17	**2.12**
RTMPose-l	384x288	Body8	17	2.13
RTMPose-m	256x192	Body8	17	2.22
RTMPose-m	384x288	Body8-Halpe26	26	2.23
RTMPose-l	256x192	Body8	17	2.27
RTMPose-m	256x192	Body8-Halpe26	26	2.28
HRNet-w32	256x192	COCO	17	2.34
RTMPose-l	384x288	Body8-Halpe26	26	2.42
RTMPose-l	256x192	Body8-Halpe26	26	2.43
RTMPose-s	256x192	Body8	17	2.52
RTMPose-s	256x192	Body8-Halpe26	26	2.56
RTMPose-l	256x192	COCO-WholeBody	133	2.59
RTMPose-m	256x192	COCO-WholeBody	133	2.61
RTMPose-t	256x192	Body8-Halpe26	26	2.68
SimCC_ViPNAS-MBNetv3	256x192	COCO	17	2.70
RTMPose-t	256x192	Body8	17	2.70
Yoloxpose_m	640x640	COCO	17	2.74

Table 6.1: Evaluation results on the custom bike fitting dataset, using only 6 keypoints most important to bikefitting (camera facing ankle, knee, hip, shoulder, elbow and wrist). The results are sorted by the Normalized Mean Error, defined in section 6.1.4. It is an average distance between the predicted keypoints and the ground truth keypoints, normalized by the average bounding box dimension, expressed as a percentage. The larger RTMPose models trained on the Body8 datasets outperform the other models.

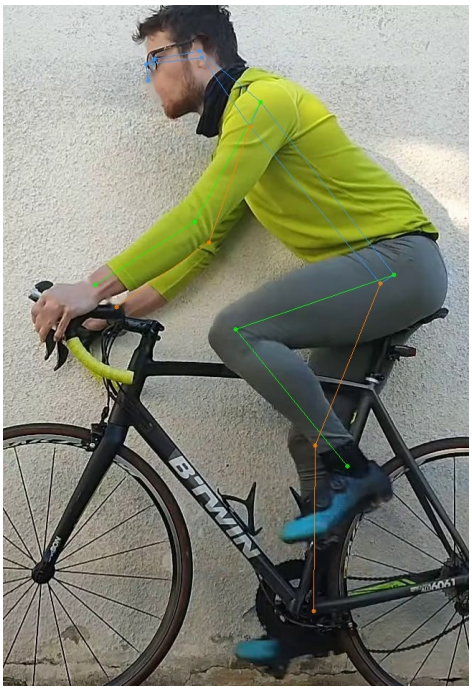
and heel are also evaluated. The results are shown in table 6.2. We can see, that the Normalized Mean Error (NME) for the foot and heel keypoints is much higher than for the other keypoints. This highlights the difficulty of predicting the foot and heel keypoints, as these keypoints are less represented in the training datasets. The models that output the WholeBody keypoints perform slightly worse than the models that only output Halpe26 keypoints, but the difference is smaller than when comparing only the 6 keypoints. What is surprising, is that the COCO-WholeBody models perform better than the Halpe26 on the foot and heel keypoints. This can be because the COCO-WholeBody dataset keypoints may match the foot and heel keypoints defined in the bikefitting better than the Halpe26 dataset. However, overall the best performing models are still the RTMPose models trained on the Body8 dataset, with the RTMPose-m 384x288 model achieving the best results with a mean NME of **2.49%**.

Architecture	Input Size	Trained On	Keypoints	Foot (%)	Heel (%)	Mean (%)
RTMPose-m	384x288	Body8-Halpe26	26	3.45	3.40	**2.49**
RTMPose-m	256x192	Body8-Halpe26	26	3.29	3.62	2.54
RTMPose-l	256x192	Body8-Halpe26	26	3.56	3.21	2.64
RTMPose-l	384x288	Body8-Halpe26	26	3.48	3.58	2.67
RTMPose-l	256x192	COCO-WholeBody	133	**2.55**	3.45	2.68
RTMPose-s	256x192	Body8-Halpe26	26	3.14	3.52	2.73
RTMPose-m	256x192	COCO-WholeBody	133	3.25	**3.15**	2.74
RTMPose-t	256x192	Body8-Halpe26	26	4.21	3.20	2.91

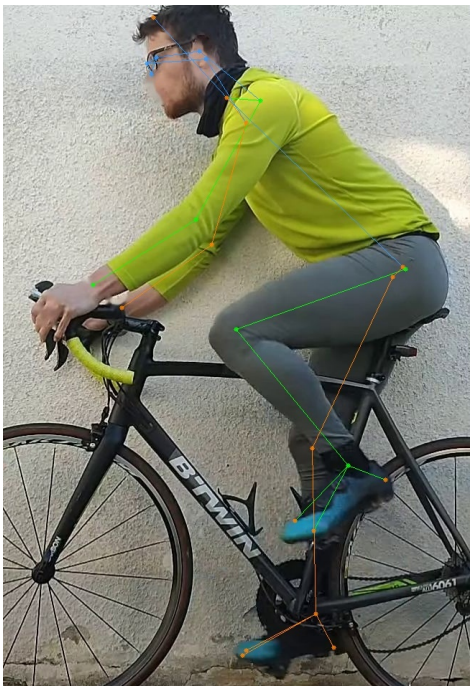
Table 6.2: Evaluation results for the models with extended keypoint set on the custom bike fitting dataset, including the heel and foot landmarks. Results for the other keypoints can be found in table 4.1. Foot column shows the Normalized Mean Error (NME) for the foot landmark, Heel column shows the NME for the heel landmark and Mean column shows the average NME for all of the 8 keypoints present in the bikefitting dataset. The values are represented as a percentage of the average bounding box dimension for a given frame. Details of the NME metric are described in section 6.1.4.

Figure 6.2 shows example predictions of the models on the bikefitting dataset. The RTMPose-m 384x288 Body8 model achieves the best results. The RTMPose-m 384x288 Body8-Halpe model also achieves good results, and also predicts the heel and foot keypoints. The RTMPose-l 256x192 COCO-WholeBody model performs much worse on the foot and heel keypoints. The accuracy of the SimCC_ViPNAS-MBNetv3 is also worse than the RTMPose models.

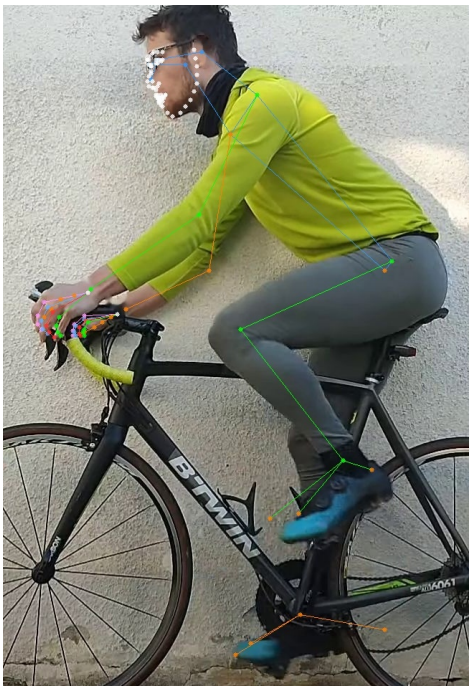
The RTMPose models show good results on the bikefitting dataset. However, the accuracy on the foot and heel keypoints is still not satisfactory even for the largest models trained on the Body8-Halpe26 dataset. The results are even worse for the models trained on the COCO-WholeBody dataset. To improve the accuracy of the models on the foot and heel keypoints, the models need to be fine-tuned on the bikefitting dataset. The details of the fine-tuning are described in the next section.



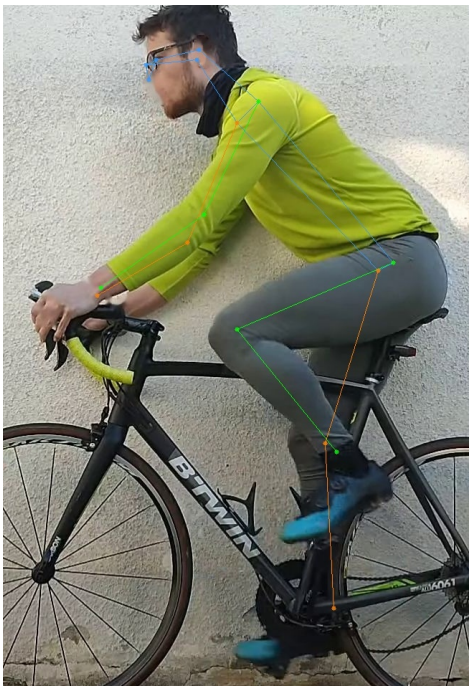
(a) RTMPose-m 384x288 Body8



(b) RTMPose-m 384x288 Body8-Halpe



(c) RTMPose-l 256x192 COCO-WholeBody



(d) SimCC_ViPNAS-MBNetv3

Figure 6.2: Example predictions of the models on the bikefitting dataset. Models (a) and (d) predict standard 17 COCO keypoints, while model (b) predicts 26 Halpe keypoints and model (c) predicts 133 WholeBody keypoints. The RTMPose-m 384x288 Body8 model (a) achieves the best results.

6.3 Fine-Tuning On Custom Bikefit Dataset

For the fine-tuning, the RTMPose-m 256x192 Body8-Halpe26, RTMPose-l 256x192 Body8-Halpe26 and RTMPose-m 384x288 Body8-Halpe26 models are selected. These models are selected, because they achieved good results on the bikefitting dataset, predict the heel and foot keypoints and are still relatively fast. The models are fine-tuned on the bikefitting dataset using the MMPose framework [10].

To train a model using the MMPose framework, the model needs to have a custom config file for the dataset. The config file contains the paths to the dataset, the model architecture, the optimizer, the learning rate scheduler and other hyperparameters. The original config files for the RTMPose models were used as a starting point and modified to use the bikefitting dataset different hyperparameters and evaluation metrics.

Custom datasets in MMPose are easiest to define using the COCO format. The dataset is defined in a JSON file, which contains the paths to the images, the annotations and the categories. The annotations contain the bounding box of the person and the keypoints. The keypoints are defined as a list of x and y coordinates and a visibility flag. The visibility flag is set to 2 if the keypoint is not visible, 1 if the keypoint is occluded and 0 if the keypoint is visible. MMPose internally uses only annotated keypoints (visibility flag 2 or 1) for training and evaluation and discards keypoints with visibility flag 0. The training set has all of the keypoints defined as annotated (visibility flags 1 or 2), so the models are trained on all of the keypoints. This ensures that the models do not unlearn the other keypoints. The validation set has only the 8 keypoints annotated by colored markers defined as annotated, so the models are evaluated only on these keypoints.

6.3.1 Training Procedure

The RTMPose models originally used the AdamW [24] optimizer with base learning rate of 4e-3, linear warmup learning rate scheduler for first 1000 iterations and cosine annealing learning rate scheduler. The models were in total trained for 700 epochs with training batch size of 512. These are tuned for a large scale dataset like COCO. For the bikefitting dataset, smaller learning rate and batch size are used. The models are trained with the AdamW optimizer with base learning rate of 5e-4, no learning rate schedulerers were used as this slowed down the training. The batch size was reduced to 16, as this sped up the training significantly. The models were trained for 130 iterations, which corresponds to 1 epoch on the bikefitting dataset. Using larger batch sizes of 32 and 64 resulted in much slower training, where the models needed 5-10 epochs of training and still achieved worse results then when using batch size of 16. The models were fast enough to be trained on an single Intel i7-7500U CPU.

Figure 6.3 shows the loss of the models during the fine-tuning. The RTMPose large and medium models with 256x192 input size achieve basically the same loss throughout the training. The RTMPose-m 384x288 model has significantly lower loss than the other models throughout the whole training. Since its validation results are on par with the 256x192 models (as discussed in the next section), it is probably because different parameters are used to construct the ground truth heatmaps for the 384x288 model, which results in lower loss. Overall the loss of the models is decreasing throughout the training, at first rapidl and then about step 60 more slowly. However, experiments showed that while the loss is not decreasing very fast, the results are still improving.

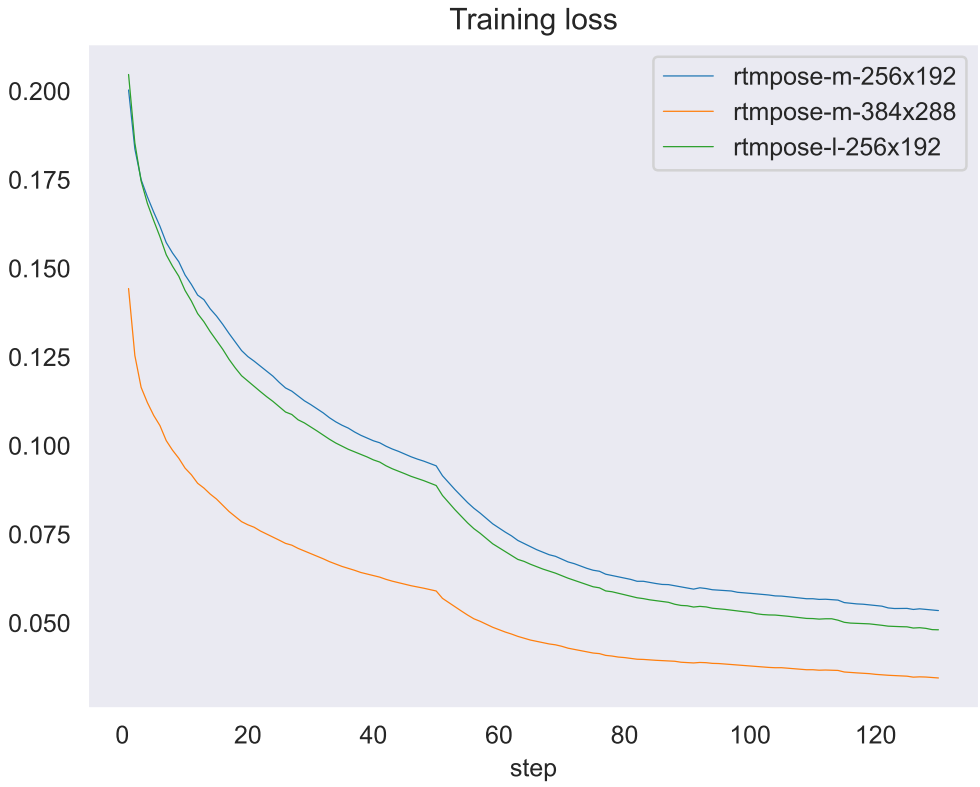


Figure 6.3: Loss of the RTMPose models during fine-tuning on the bikefitting dataset. The models were trained for 130 iterations, which corresponds to 1 epoch on the bikefitting dataset. For optimization, AdamW optimizer with base learning rate of 5e-4 and batch size of 16.

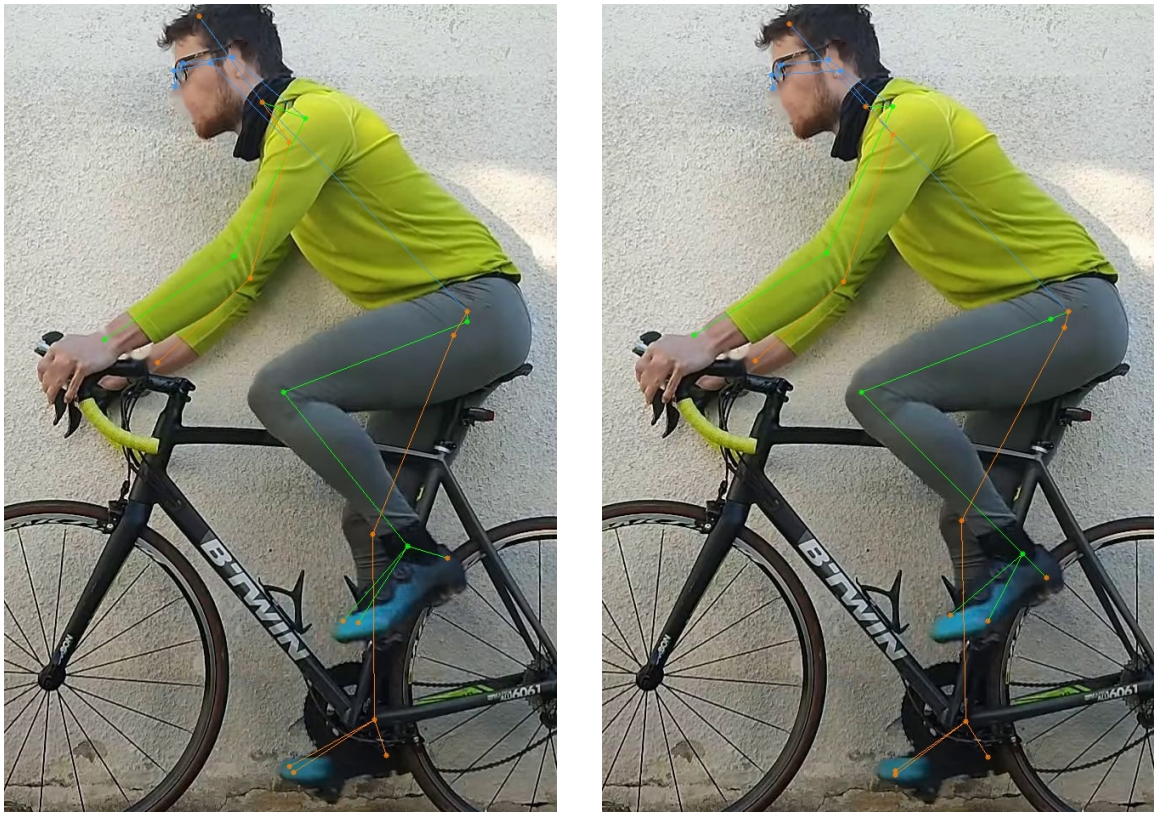
6.4 Evaluation Of Fine-Tuned Models

Table 6.3 shows the results of the fine-tuned models on the bikefitting dataset. The models are evaluated using the NME metric, described in section 6.1. All of the fine-tuned models achieve significantly better results than the pre-trained models. Despite the differences in the performance of medium and large pre-trained models, the fine-tuned models achieve similar results. The best results are achieved by the RTMPose-m 384x288 model, which achieves a mean NME of **1.50%**, but the other fine-tuned models achieve very similar results. The fine-tuned models achieve a significant improvement in the accuracy of the foot and heel keypoints. The foot and heel keypoints are still the most difficult to predict, but the fine-tuned models achieve much better results than the pre-trained models. Because there is not a significant difference in the performance of the fine-tuned models, the RTMPose-m 256x192 model is selected for the bikefit application, as it is the smallest and fastest model that achieves good results.

Architecture	Input Size	Trained On	Keypoints	Foot (%)	Heel (%)	Mean (%)
RTMPose-m	384x288	Bikefit dataset	26	**1.25**	2.11	**1.50**
RTMPose-m	256x192	Bikefit dataset	26	1.45	1.99	1.51
RTMPose-l	256x192	Bikefit dataset	26	1.26	**1.94**	1.52
RTMPose-m	384x288	Body8-Halpe26	26	3.45	3.40	2.49
RTMPose-m	256x192	Body8-Halpe26	26	3.29	3.62	2.54
RTMPose-l	256x192	Body8-Halpe26	26	3.56	3.21	2.64

Table 6.3: Evaluation results for the fine-tuned models on the custom bike fitting dataset. The models are fine-tuned on the bikefitting dataset for 130 iterations. The results are sorted by the Normalized Mean Error, defined in section 6.1.4. It is an average distance between the predicted keypoints and the ground truth keypoints, normalized by the average bounding box dimension, expressed as a percentage. All of the fine-tuned models achieve significantly better results than the pre-trained models.

Figure 6.4 shows example predictions of the RTMPose-m 256x192 model before and after fine-tuning. The fine-tuned model achieves much better results on the foot and heel keypoints. It also shifts the knee and shoulder keypoints more closely to the location which are used for example in the Retul bikefitting system. However, it also shifts the hip keypoint closer to the knee keypoint, which is not desirable. Similarly, the wrist keypoint is now no longer predicted on the center of rotation of the wrist joint, but more upwards. These changes are probably caused by misplacement of the colored markers, which is especially hard to avoid when muliple poses and camera angles are present in the dataset. Nevertheless, the fine-tuned model overall achieves better results than the pre-trained model and is used in the bikefit application.



(a) RTMPose-m 256x192 Body8-Halpe26

(b) RTMPose-m 256x192 Body8-Halpe26 Fine-Tuned

Figure 6.4: Example predictions of the RTMPose-m 256x192 model before and after fine-tuning on the bikefitting dataset. The fine-tuned model achieves much better results on the foot and heel keypoints, but also undesirably shifts the hip and wrist keypoints, probably due to misplacement of the colored markers.

Chapter 7

Bikefit Web Application

This chapter describes the web application for bike fitting. Implementing the application as a web application has several advantages over implementing it as a mobile or desktop application. The web application can be used on any device with a web browser. This allows the application to be used on mobile devices, tablets, laptops and desktop computers. The user also does not need to download and install anything.

Most web applications are implemented using the client-server architecture with the video processing and other more complicated tasks (such as pose estimation) being performed on the server. This has several disadvantages. The server needs to be powerful enough to handle the load of multiple users. This can be quite expensive for heavy models and a lot of users. The user data also needs to be sent to the server, which means the user needs to be online when using the application and the users can be concerned about their privacy.

To avoid these disadvantages, most of the bussiness logic of the application (including video processing, keypoint estimation and generation of the suggestions) is implemented on the client side.

The application is implemented using the Svelte framework[1] . Svelte is a component framework similar to React[2] or Vue[3]. It is used to create reactive user interfaces. Svelte compiles the application to vanilla JavaScript, which means the application does not need to include a large framework library. This makes the application smaller and faster. Svelte also has a built-in state management system, which is used to store the application state.

To process the video frame by frame, the application uses the WebCodecs API. The details of the implementation are described in section 7.2.

To estimate the keypoints, multiple approaches were tested. First approach uses colored markers, which are placed on the parts of the body, whose position needs to be estimated (foot, heel, ankle, knee, hip, shoulder, elbow, wrist). The markers are then detected using color thresholding and tracked using a method similar to SORT. This approach is implemented using the OpenCV.js library [34]. The details of the implementation are described in section 3.

The second approach uses the RTMPose pose estimation model and a person detector. The models run in the browser using the Tensorflow.js library[4]. The models are described in section 4.1 and evaluated in chapter **??**. To improve the performance of the pose estimation

[1] https://svelte.dev/
[2] https://react.dev/
[3] https://vuejs.org/
[4] https://www.tensorflow.org/js

models on the task of side view pose estimation of cyclists, the models are trained on a custom dataset. The details of the training are described in section 6. The details of the implementation are described in section 7.3.

7.1 Application Structure

From the user's perspective, the application consists of three main steps.

1. **Video Upload** The first step is the video upload. The user is prompted to upload a video of the cyclist from the side view. The user can also choose to use a demo video for testing the application. After the video is uploaded, the user can proceed to the next step. Figure 7.1 shows the first step of the application.

2. **Select Fit Goals** On the second step, the user can review and play the uploaded video and is asked to select the type of the bike (road, gravel and mountain bike) and the fit goals (comfort, performance, balanced). These fit goals and the type of the bike are used to adjust the suggestions for the bike fit. After the fit goals are selected, the user can process the video by the pose estimation model. Figure 7.2 shows the second step of the application. While the video is processing the user can see the progress of the processing by the progress circle in the middle of the screen. Figure 7.3 shows the processing of the video.

3. **Examine Suggestions** On the last step, after the video is processed, the user can examine the keypoint predictions from the pose estimation model, computed joint angles, suggested angle ranges and the suggestions for the bike fit. The user can also play the video with the keypoint predictions overlaid on the video. Figure 7.4 shows the first part of the last step of the application and figure 7.5 shows the second part of the last step of the application.

7.2 Video Upload and Processing

At first, the video needs to be uploaded to the application. The video is uploaded using the HTML5 file input. The file is then converted to a URL using the `URL.createObjectURL()` method, because the WebCodecs API requires a URL as input.

7.2.1 Accessing the Video Frames

To accurately estimate the keypoints, the application needs to access all of the video frames and process them. This is not trivial to do in the browser. The browser does not allow the application to access the video frames directly.

To process the video frame by frame, the application uses the WebCodecs API[5]. The WebCodecs API is a low-level API for encoding and decoding audio and video. It gives developers access to the invididual frames of the video. It is the only way to access the frames of the video in the browser. Simple and commonly used method is to simultaneously play the video and draw the video frames to a canvas element and then access the pixels of the canvas. However, this method only does not work well if the processing is computationally

[5] https://developer.mozilla.org/en-US/docs/Web/API/WebCodecs_API

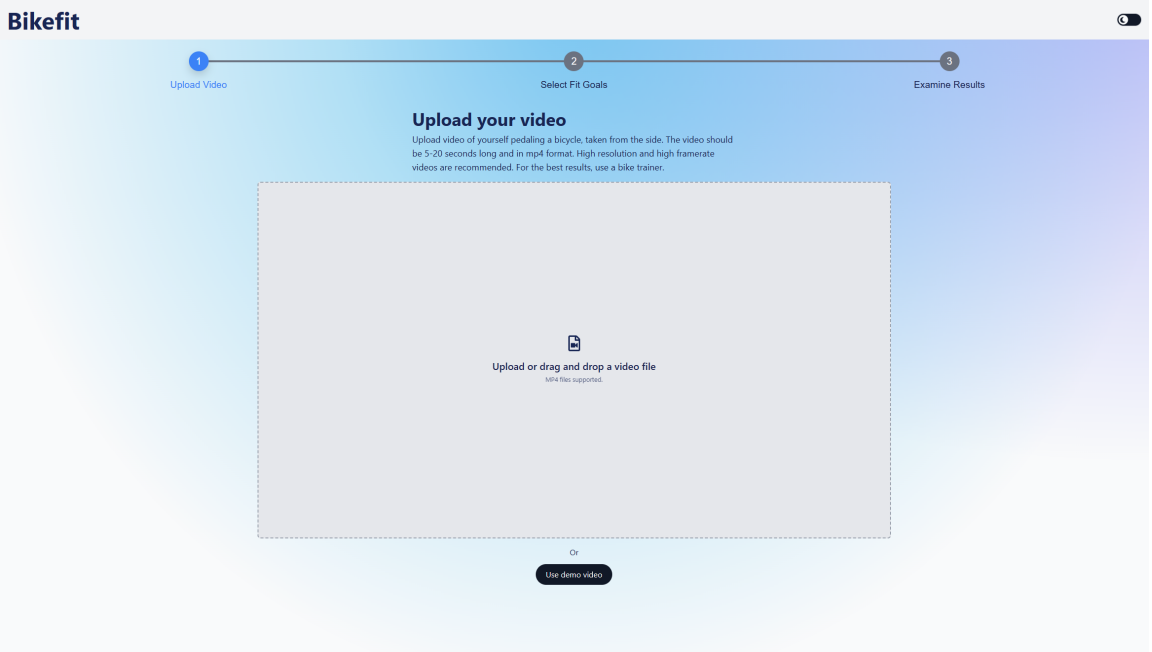


Figure 7.1: The first step of the bikefit application. The user is prompted to upload a video of the cyclist from the side view. The user can also choose to use a demo video for testing the application. After the video is uploaded, the user can proceed to the next step.

expensive, since the video is played in real time. This leads to skipping of the frames and not all of the frames are processed. The WebCodecs API allows the application to process the video at its own pace.

Because the API is relatively new, it is not yet supported by all browsers. Figure 7.6 shows the browser support for the WebCodecs API as of January 2024. The API is supported by all browsers based on Chromium (Google Chrome, Microsoft Edge, Opera, Brave, etc.). It is Partially supported by Safari and Safari on iOS (video only) and not supported by Firefox. Overall based on CanIUse data[6], the API is supported by 87.6% of the users, where 74.5% of the users have full support and 13.1% of the users have partial support.

Due to the low-level design of the WebCodecs API, it is not easy to use. Moreover, the API is quite new and not very well documented so there is not a lot of examples. The developer also needs to take care of demuxing the media containers. This work uses the getVideoFrames.js library[7] from josephrocca. The library internally uses the WebCodecs API and MP4Box.js[8] for demuxing mp4 files, but provides a simple interface for accessing the video frames. The library is used to get the video frames and then the frames are processed as described in sections 3 and 7.3. It defines three callback functions. The `onFrame` callback is called for each frame of the video. The `onConfig` callback is called when the video metadata is loaded. The `onFinish` callback is called when all of the frames are processed. Overall, WebCodecs API is a powerful tool for processing video in the browser, but it relies on third party demuxing libraries and is not yet supported by all browsers.

[6] https://caniuse.com/webcodecs
[7] https://github.com/josephrocca/getVideoFrames.js
[8] https://github.com/gpac/mp4box.js/

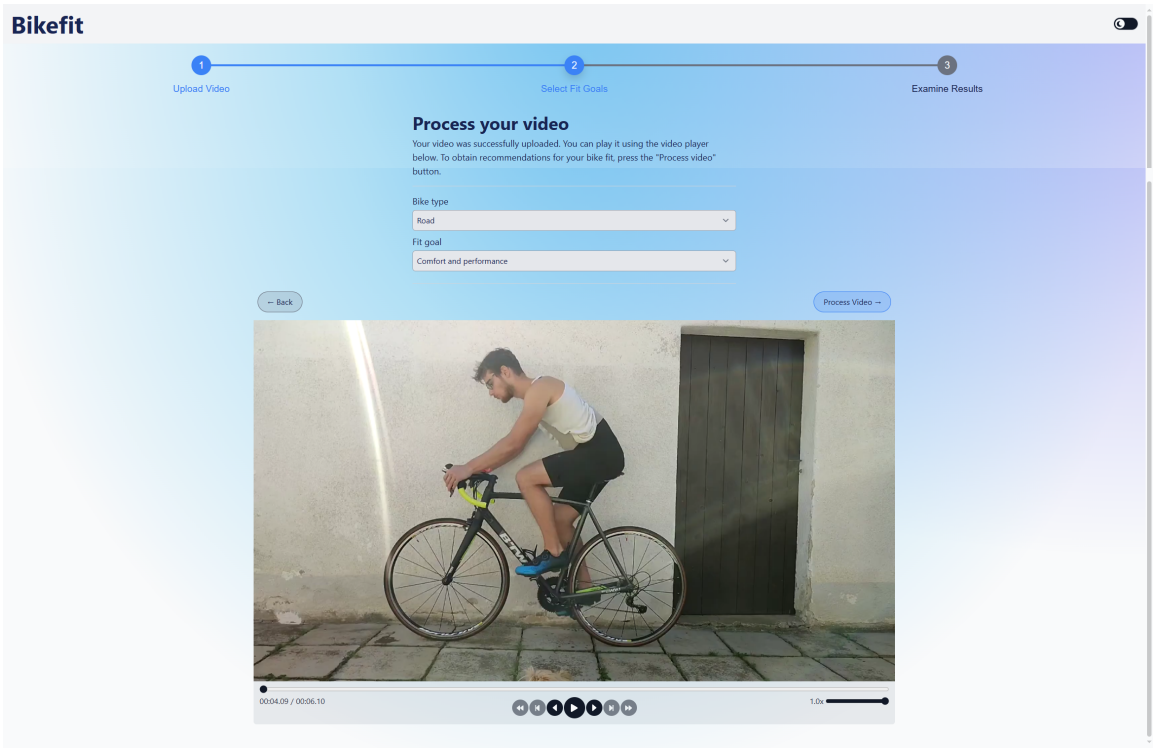


Figure 7.2: The second step of the bikefit application. The user can review and play the uploaded video and is asked to select the type of the bike (road, gravel and mountain bike) and the fit goals (comfort, performance, balanced). After the fit goals are selected, the user can process the video by the pose estimation model. The video speed can be adjusted by the slider on the right. The user can also seek forward and backward 1 frame by clicking the buttons on the left and right of the middle play button.

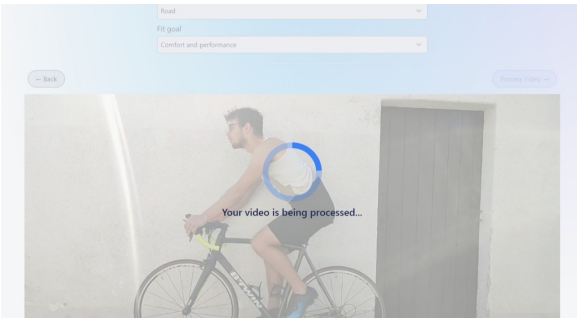


Figure 7.3: After the user selects the fit goals, the video is processed by the pose estimation model. The user can see the progress of the processing by the progress circle in the middle of the screen.

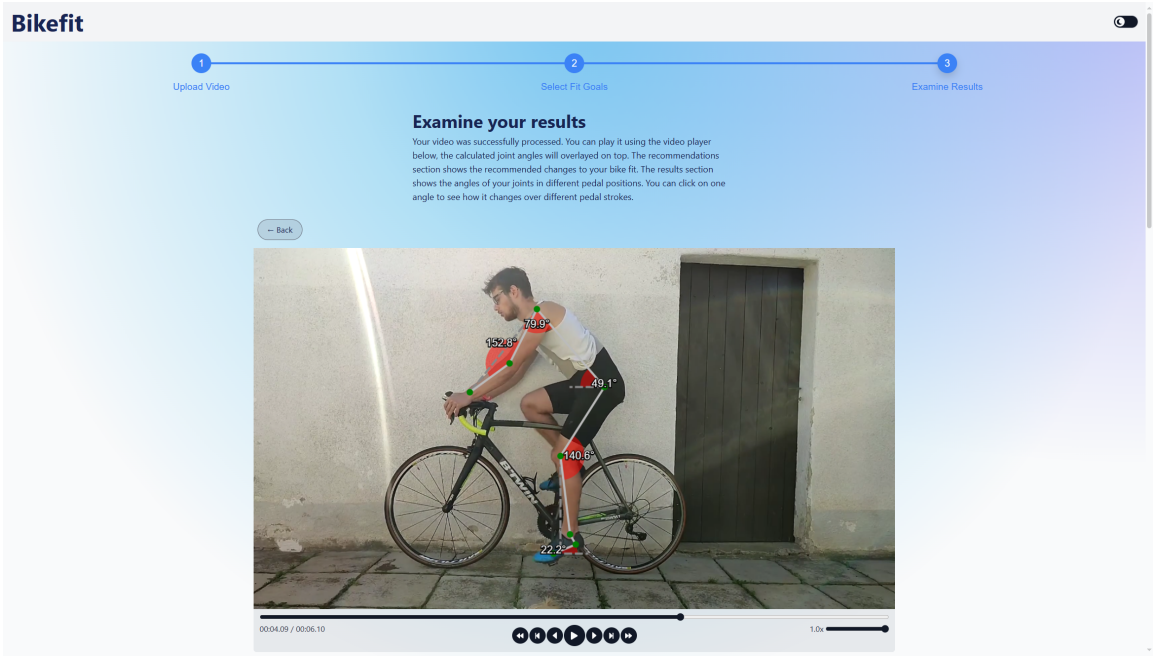


Figure 7.4: The first part of the last step of the bikefit application. The user can play the video with the keypoint predictions and measured angles overlaid on the video. The video speed can be adjusted by the slider on the right. The user can also seek forward and backward 1 frame by clicking the buttons on the left and right of the middle play button. The buttons on the side can be used to seek to the next or previous pedal position and next or previous pedal stroke.

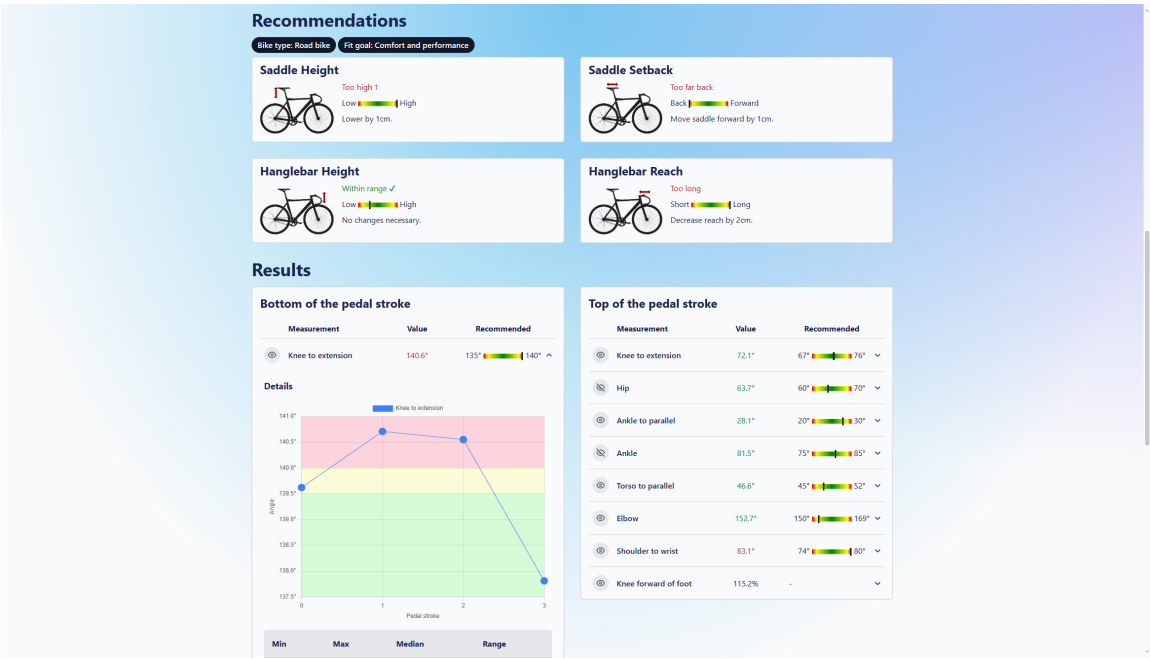


Figure 7.5: The second part of the last step of the bikefit application. In the „Recommendations“ section, the user can see the suggested changes to their saddle height, saddle setback, handlebar height and handlebar reach. In the „Results“ section, the user can see the recommended ranges and measured values for the angles and measurements in each of the pedal positions (bottom, top and front (not shown)). After clicking on a measurement, the user can see a chart with the angle values for each pedal stroke, with sections of the chart colored according to the recommended ranges.

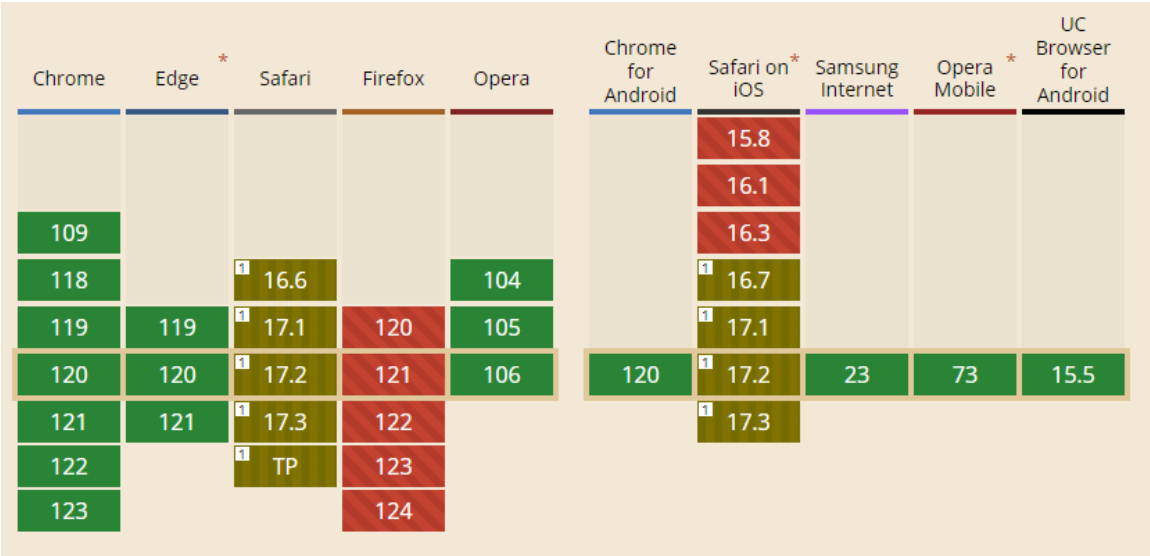


Figure 7.6: Browser support as of January 2024 for the WebCodecs API. Green color indicates full support, yellow color indicates partial support and red color indicates no support. Taken from https://caniuse.com/webcodecs.

7.3 Pose Estimation

To get the locations of the 8 keypoints (foot, heel, ankle, knee, hip, shoulder, elbow, wrist) in the video, the RTMPose-m 256x192 Body8-Halpe26 model is used. The model is trained on the custom bike fitting dataset, as described in chapter 6. The model is run on each frame of the video and the keypoints are extracted.

Since JavaScript is single-threaded and the model is computationally expensive, the model is run in a Web Worker. Web Worker is a JavaScript feature that allows the application to run a script in a separate thread. This allows the application to run long tasks in the background without blocking the main thread.

The keypoints are then used to compute the joint angles and generate the suggestions for the bike fit.

7.3.1 Model Conversion and Deployment

The RTMPose models are originally trained in the PyTorch framework. To run the model in the browser, the model needs to be converted to either ONNX or TensorFlow.js format. ONNX model can be run in the browser using ONNX Runtime Web [9]. However, the ONNX Runtime Web only provides API for running the model, but does not provide tensor operations needed for preprocessing the input and postprocessing the output. Due to these reasons, TensorFlow.js is used instead.

To convert the model to TensorFlow.js format, the model is first converted to ONNX format using the MMDeploy [9] toolset. The MMDeploy toolset is a set of tools for converting models from the various OpenMMLab frameworks (MMPose, MMAction, MMDetection, etc.) to ONNX and other formats and provides an easy way to deploy the RTMPose models.

The ONNX model is then first converted to the TensorFlow SavedModel format using the unofficial onnx2tf [15] tool. The SavedModel format is a universal format for storing TensorFlow models. The SavedModel format is then converted to the TensorFlow.js format using the official TensorFlow.js converter [10] provided by the TensorFlow.js team.

The TensorFlow.js converter also quantizes the model to float16 to reduce the size of the model and speed up the inference. Quantization is a process of converting the weights and activations of the model from float32 to float16. This reduces the size of the model by half and can speed up the inference without significant loss of accuracy. The quantized model is then used in the application.

TensorFlow.js provides several different backends for running the model. The CPU backend runs the model on CPU, using JavaScript for the computation and is the slowest, but has support for all of the operations. The WebGL backend runs the model on the GPU using WebGL shaders and is the faster, but introduces very noticeable latency when the model is first run, because the WebGL API is not designed for running deep learning models and the model needs to be compiled to the shaders. The WebAssembly backend runs the model on the CPU using WebAssembly and is the faster than the CPU backend, but slower than WebGL for the larger models. The WebGPU backend runs the model on the GPU and should be the fastest, because it is designed with deep learning in mind, but is not yet supported by all browsers. From the major browsers, the WebGPU backend is only supported by Google Chrome, Microsoft Edge and Chrome on Android. This application uses the WebGPU backend if it is available, otherwise it falls back to the WebGL backend.

[9] https://onnxruntime.ai/docs/tutorials/web/

[10] https://github.com/tensorflow/tfjs/tree/master/tfjs-converter

7.3.2 Preprocessing

The RTMPose models are a top-down pose estimation model, which means that the model requires the input image to be cropped around the person. To crop the image, the bounding box of the person is first detected using the COCO-SSD [11] model, available in the TensorFlow.js model zoo. It is a Single Shot Detector (SSD) [23] model trained on the COCO [22] dataset. To speed up processing the model is ran only once on the first frame of the video, because the person is assumed to be in the same position in all of the frames. The COCO-SSD is not the most accurate or the fastest model, but it is readily available in the TensorFlow.js model zoo and is good enough for using it on the first frame of the video.

The bounding box of the person is then expanded by 2.5% on the left and right and by 20% on the bottom to account for changing legs position of the user due to pedaling. The bounding box is then used to crop the image. The cropped image is then resized to 256x192 pixels, which is the input size of the RTMPose model and normalized by the mean and standard deviation of the Body8 dataset (mean = [123.675, 116.28, 103.53], std = [58.395, 57.12, 57.375]).

7.3.3 Postprocessing

The RTMPose models output a SimCC [20] heatmap, which is a heatmap of the keypoints where the x and y axis are separated. This significantly reduces the number of parameters of the model and speeds up the inference. For each keypoint, the model outputs a list of scores for x and y axis. To find the location of the keypoint, the argmax of x and y axis is taken. The x and y coordinates are then divided by 2 because the models output the coordinates multiplied by 2 to increase accuracy. The x and y coordinates are then adjusted for the resizing and added to the bounding box coordinates to get the coordinates in the original image. Only 8 body keypoints on the camera facing side (foot, heel, ankle, knee, hip, shoulder, elbow, wrist) are used further, the other keypoints are discarded. The keypoints over all of the frames are then smoothed usign a Savitzky-Golay filter [32] to reduce the noise in the predictions.

7.4 Computing Joint Angles and Measurements

After the location of the 8 keypoints is estimated, the joint angles and measurements for the current frame are computed. Joint angles are specified either by three keypoints if the angle is computed between two bones or by two keypoint if the angle is computed between one bone and the horizontal axis.

Other than joint angles, application also computes the knee forward of foot measurement, which is the distance between the knee and the vertical line through the foot. Due to the absence of real world units in the video, the measurement is expressed as the percentage of the length from the fifth metatarsal to the heel. This measurement is used to determine saddle setback as its very similar to the widely used knee over pedal spindle (KOPS) measurement.

[11] https://github.com/tensorflow/tfjs-models/tree/master/coco-ssd

7.4.1 Determining the Pedal Position

For most of the measurments, the values for every frame are not in themselves very useful. For measurements that barely change over the pedal stroke (mainly upper body joint angles), the average value over all of the frames is used. For measurements that change over the pedal stroke (lower body measurements such as knee angle, knee forward of foot, ankle angle), are compared to the recommended ranges in a specific pedal position (bottom, top, front). Therefore the current pedal position needs to be determined.

To determine the pedal position for each frame, invididual pedal strokes need to be detected. To do that, a circle is fitted to the foot keypoints over all of the frames using a least squares method. This circle is then used to determine the pedal position. For example, to determine the front pedal position, the angle of the foot keypoint on the circle is computed for each frame in a pedal stroke. The front pedal position is then determined as the frame with the angle closest to 90 degrees (horizontal forward). Similarly, the bottom pedal position is determined as the frame with the angle closest to 180 degrees (vertical down) and the top pedal position is determined as the frame with the angle closest to 0 degrees or 360 degrees (vertical up). After obtaining the list of frames for each pedal position, and list of frames for each pedal stroke, we can finally compute the measurements and joint angles for each pedal position. To do this, median value of the measurements for each pedal position is used. This is done to avoid the influence of the outliers in the measurements.

7.5 Generating Suggestions

After the joint angles and measurements are computed for each pedal position, the suggestions for the bike fit are generated. The suggestions are generated based on comparing some of the measurements in the pedal positions to the recommended ranges. The recommended ranges are based on the bike fit guidelines and are then adjusted based on the type of the bike and the fit goals selected by the user. For example, saddle height recommendations are based on knee angle at the bottom of the pedal stroke. If the knee angle is inside the recommended range, saddle height is considered correct. If the knee angle at the bottom of the pedal stroke is at the lower 10 percent of the recommended range, a suggestion to increase the saddle height slightly (by 1 cm) is generated. If the knee angle is lower than the recommended range, a suggestion to increase the saddle height significantly (by 2 cm) is generated. Suggestions are made similarly when the knee angle is at the upper 10 percent of the recommended range or higher than the recommended range.

In total, suggestions are generated for the saddle height, saddle setback, handlebar height and handlebar reach. As was mentioned before, saddle height is determined by the median knee angle at the bottom of the pedal stroke. Saddle setback is determined by the median knee forward of foot measurement at the front of the pedal stroke. Handlebar height is determined by the median shoulder angle at all of the frames and handlebar reach is determined by the median shoulder angle at all of the frames.

Basing suggestions only on a single measurement is not ideal, because in reality the bike fit is a complex process and the measurements are interdependent. For example, increasing the saddle height will also increase the saddle setback, handlebar reach also influences back angle not only shoulder angle and so on. In the future, the suggestions could be generated using a more complex model, which would take into account all of the measurements and their interdependencies.

7.6 Video Player

To play the video with the predicted keypoints and measurements overlaid on the video, the application implements a custom video player based on the HTML canvas and video elements. The video is played on a hidden video element and the frames are drawn to a canvas element. This is done so that joint angles and measurements can be drawn on top of the video frames. The video player allows the user to play the video, seek to a specific time, seek to the next or previous pedal position and seek to the next or previous pedal stroke. The video player also allows the user to adjust the speed of the video.

However, there does not exist a standard way to get the index of the current frame of the video. The video element has a `currentTime` property, which returns the current time of the video in seconds, but there is no property that returns the current frame index, or even the video frames per second. Another reason why relying on the `currentTime` property is not ideal is that the `currentTime` property returns the time of the video but the video can be longer than the video track, also the video track can have variable frame rate.

This is why, to get the current frame index, the application saves the frame timestamps obtained when processing the video with the WebCodecs API. And then uses the requestVideoFrameCallback [12] to get the current frame index. The requestVideoFrameCallback is a new API that allows the application to accurately get the current video track time using the `metadata.mediaTime` property. The `metadata.mediaTime` property returns the current time of the video track in seconds. The application then finds the closest frame timestamp to the current video track time and uses the index of the frame timestamp as the current frame index. This way the application can accurately get the current frame index of the video.

7.7 Presentation of the Results and Recommendations

[12] https://developer.mozilla.org/en-US/docs/Web/API/HTMLVideoElement/requestVideoFrameCallback

Chapter 8

Experiments

Chapter 9

Conclusion

Bibliography

[1] Andriluka, M., Iqbal, U., Insafutdinov, E., Pishchulin, L., Milan, A. et al. PoseTrack: A Benchmark for Human Pose Estimation and Tracking. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[2] Andriluka, M., Pishchulin, L., Gehler, P. and Schiele, B. 2D Human Pose Estimation: New Benchmark and State of the Art Analysis. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[3] Bertalmio, M., Bertozzi, A. and Sapiro, G. Navier-stokes, fluid dynamics, and image and video inpainting. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. 2001, vol. 1, p. I–I. DOI: 10.1109/CVPR.2001.990497.

[4] Bewley, A., Ge, Z., Ott, L., Ramos, F. and Upcroft, B. Simple online and realtime tracking. In: IEEE. *2016 IEEE international conference on image processing (ICIP)*. 2016, p. 3464–3468.

[5] Bike Fast Fit. *Bike Fast Fit Elite v2 - Basic Bike Fitting*. 2023. Available at: https://www.youtube.com/watch?v=4xhFhvIM7R4.

[6] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 2000.

[7] Burt, P. *Bike Fit 2nd Edition: Optimise Your Bike Position for High Performance and Injury Avoidance*. Bloomsbury Publishing, 2022.

[8] Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y. et al. MMDetection: Open MMLab Detection Toolbox and Benchmark. *ArXiv preprint arXiv:1906.07155*. 2019.

[9] Contributors, M. *OpenMMLab's Model Deployment Toolbox*. [https://github.com/open-mmlab/mmdeploy]. 2021.

[10] Contributors, M. *OpenMMLab Pose Estimation Toolbox and Benchmark* [https://github.com/open-mmlab/mmpose]. 2020.

[11] Fang, H.-S., Li, J., Tang, H., Xu, C., Zhu, H. et al. AlphaPose: Whole-Body Regional Multi-Person Pose Estimation and Tracking in Real-Time. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2022.

[12] Ge, Z., Liu, S., Wang, F., Li, Z. and Sun, J. Yolox: Exceeding yolo series in 2021. *ArXiv preprint arXiv:2107.08430*. 2021.

[13] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B. et al. Searching for mobilenetv3. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, p. 1314–1324.

[14] Hua, W., Dai, Z., Liu, H. and Le, Q. Transformer quality in linear time. In: PMLR. *International Conference on Machine Learning*. 2022, p. 9099–9117.

[15] Hyodo, K. *Tools to convert ONNX files (NCHW) to TensorFlow format (NHWC)*. Available at: https://github.com/PINTO0309/onnx2tf.

[16] Jhuang, H., Gall, J., Zuffi, S., Schmid, C. and Black, M. J. Towards understanding action recognition. In: *International Conf. on Computer Vision (ICCV)*. December 2013, p. 3192–3199.

[17] Jiang, T., Lu, P., Zhang, L., Ma, N., Han, R. et al. RTMPose: Real-Time Multi-Person Pose Estimation based on MMPose. *ArXiv preprint arXiv:2303.07399*. 2023.

[18] Jin, S., Xu, L., Xu, J., Wang, C., Liu, W. et al. Whole-Body Human Pose Estimation in the Wild. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020.

[19] Li, J., Wang, C., Zhu, H., Mao, Y., Fang, H.-S. et al. CrowdPose: Efficient Crowded Scenes Pose Estimation and a New Benchmark. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[20] Li, Y., Yang, S., Liu, P., Zhang, S., Wang, Y. et al. Simcc: A simple coordinate classification perspective for human pose estimation. In: Springer. *European Conference on Computer Vision*. 2022, p. 89–106.

[21] Lin, J., Zeng, A., Wang, H., Zhang, L. and Li, Y. One-Stage 3D Whole-Body Mesh Recovery with Component Aware Transformer. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, p. 21159–21168.

[22] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P. et al. Microsoft COCO: Common Objects in Context. In: Fleet, D., Pajdla, T., Schiele, B. and Tuytelaars, T., ed. *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014, p. 740–755. ISBN 978-3-319-10602-1.

[23] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. et al. SSD: Single Shot MultiBox Detector. In: Leibe, B., Matas, J., Sebe, N. and Welling, M., ed. *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, p. 21–37. ISBN 978-3-319-46448-0.

[24] Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *ArXiv preprint arXiv:1711.05101*. 2017.

[25] Lyu, C., Zhang, W., Huang, H., Zhou, Y., Wang, Y. et al. Rtmdet: An empirical study of designing real-time object detectors. *ArXiv preprint arXiv:2212.07784*. 2022.

[26] MacQueen, J. et al. Some methods for classification and analysis of multivariate observations. In: Oakland, CA, USA. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. 1967, vol. 1, no. 14, p. 281–297.

[27] Maji, D., Nagori, S., Mathew, M. and Poddar, D. Yolo-pose: Enhancing yolo for multi person pose estimation using object keypoint similarity loss. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, p. 2637–2646.

[28] Retül Technology. *XY Fit Step 4: Saddle Fore-Aft*. 2021. Available at: https://www.youtube.com/watch?v=hZgI87DUUbU.

[29] Retül Technology. *XY Fit Steps 1-8: Markers and Harness*. 2021. Available at: https://www.youtube.com/watch?v=k-FsejpibKE.

[30] Ribeiro Branco, G., De Michelis Mendonça, L., Alves Resende, R. and Pivetta Carpes, F. Does the Retül System provide reliable kinematics information for cycling analysis? *Journal of Science and Cycling*. Dec. 2022, vol. 11, no. 3, p. 76–84. DOI: 10.28985/1322.jsc.15. Available at: https://www.jsc-journal.com/index.php/JSC/article/view/759.

[31] Ronchi, M. R. and Perona, P. Benchmarking and Error Diagnosis in Multi-instance Pose Estimation. In:. October 2017. DOI: 10.1109/ICCV.2017.48.

[32] Savitzky, A. and Golay, M. J. E. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry*. 1964, vol. 36, no. 8, p. 1627–1639. DOI: 10.1021/ac60214a047. Available at: https://doi.org/10.1021/ac60214a047.

[33] Sun, K., Xiao, B., Liu, D. and Wang, J. Deep High-Resolution Representation Learning for Human Pose Estimation. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[34] Taheri, S., Vedienbaum, A., Nicolau, A., Hu, N. and Haghighat, M. R. Opencv. js: Computer vision processing for the open web platform. In: *Proceedings of the 9th ACM Multimedia Systems Conference*. 2018, p. 478–483.

[35] Telea, A. C. An Image Inpainting Technique Based on the Fast Marching Method. *Journal of Graphics Tools*. 2004, vol. 9, p. 23 – 34. Available at: https://api.semanticscholar.org/CorpusID:5908881.

[36] Ultralytics. *YOLOv5: A state-of-the-art real-time object detection system* [https://docs.ultralytics.com]. 2021. Accessed: 14.01.2024.

[37] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. Attention is all you need. *Advances in neural information processing systems*. 2017, vol. 30.

[38] Wu, J., Zheng, H., Zhao, B., Li, Y., Yan, B. et al. Ai challenger: A large-scale dataset for going deeper in image understanding. *ArXiv preprint arXiv:1711.06475*. 2017.

[39] Xu, L., Guan, Y., Jin, S., Liu, W., Qian, C. et al. Vipnas: Efficient video pose estimation via neural architecture search. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, p. 16072–16081.

[40] Zhang, S.-H., Li, R., Dong, X., Rosin, P., Cai, Z. et al. Pose2Seg: Detection Free Human Instance Segmentation. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.