Move sequence detection on bouldering problems

Charles Beauville - 283103
Data Science

Robin Debalme - 282406
Data Science

Théo Patron - 284555
Comp. Science & Eng.

***Abstract*—The goal of this project was to detect the move sequence of a boulder using bouldering videos.**
We were provided a dataset of videos. In this dataset, we first selected good videos and stabilised them. We then labeled all the usable videos for further preproccessing like cropping, cutting or taking a screenshot for visualization. Afterward, we ran a pose estimation algorithm in order to get the coordinates of the body parts of the climbers. We used this data to detect the moves of the boulder problem and we found the sequence using a clustering algorithm. Finally, we implemented some visualization functions to show our results.
We were pretty impressed with our results, the program outputs the right move sequence.

I. Introduction

Bouldering is a form of climbing where the climber climbs small rock formations or artificial rock walls without the use of ropes. In 2016, the International Olympic Committee (IOC) officially approved climbing including bouldering as an Olympic sport. Bouldering "problems" have their own solution, and to successfully climb it the climber need to work it out. This solution can be seen as a move sequence the climber has to do to reach the top. A move sequence consists of the order in which the climber has to take the holds and which body part he has to use for the different holds. Working with the Swiss Olympic Climbing team lead, we set out to detect and visualize the move sequence of a climber based on a video. This can later be useful to compare holds between a succesful climb and a failed one, to label a dataset or train a more complex model. There is little previous research we could find in this field, the most relevant one was an unfinished project [1] which aims at predicting the positions of the holds on the climbing wall given a picture.

II. Dataset

A. Original

Urs Stöcker (Swiss Olympic team leader, and formerly German Olympic team leader) provided us with a dataset of bouldering videos. These videos are from between 2017 and 2021 and are recordings of German athletes during bouldering competitions. The dataset is composed of 3307 videos.

B. Preprocessing

Some videos were not stable, or of bad quality. Thus, a very important part of the project was the selection and the preprocessing. Those steps are described in the following sections.

1) Selecting the data: The first step was to remove unusable videos. By watching all the videos, we selected the static and not blurry videos. To avoid having too few videos (as almost all the videos were not static, i.e. the camera was moving), we also kept semi-static videos. We removed the videos where a person is passing in front of the camera at some moments. Finally, all the kept videos have been sorted in different folders to have one folder per boulder problem.

2) Stabilisation: As most of the remaining videos were not static, stabilisation was required to obtain a good final result. Indeed, the move sequence is in the end printed on a screenshot of the video, and if the camera is moving, the holds are not always at the same place during the video. As the dataset is hosted on Google Drive and is too heavy to be downloaded, we had to work with Colab and to perform the stabilisation with python. This has been done using the VidStab library than can be plugged-in with FFmpeg. The parameter used is 'smoothing=0' to simulate a static camera. Using OS and Subprocess libraries, we have been able to run the VidStab batch command to all the videos in all subfolders.

3) Data labeling: In order to have some metadata about each video, we had to watch the stabilized videos to fill an Excel sheet with different information. We reported the filename, the gender of the climber, its name, the success level (if the climber failed, reached the zone, or reached the top), and the competition. This allowed us to remove the videos where stabilisation failed, i.e. when the video is still too shaky. For videos with multiple climbing attempts, we also reported `time_start` and `time_end` in order to crop the video to keep only one attempt. For videos with multiple climbers in it, or with other visible persons, we also reported `crop_x`, `crop_x_side`, `crop_y` and `crop_y_side`. Indeed, as we will see later, pose estimation works better when only one person is visible. We used the following convention:

- `crop_x/y` is a percentage of what we want to keep.
- `crop_x_side/y_side` can take the following values: 'left', 'right', 'center' for x, and 'top', 'bottom', 'center' for y.

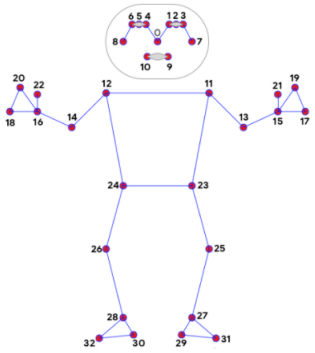
To be clearer, here are two examples:
- x: 60% right and y: 100% center means that we keep all the height but only the 60% on the right side (i.e. we remove the 40% of the image on the left).
- x: 80% center and y: 80% top means that we remove 10% on the left, 10% percent on the right (so we keep 80% of the width in the center), and we remove 20% on the bottom of the image (we keep 80% of the height).

Finally, we reported a moment of the video where we could take a screenshot in order to have an image of the boulder problem (if possible, we chose a moment when a climber is not on the wall in order to be able to see all the holds).

4) Video cropping and screenshot: In order to crop the video in time and to take a screenshot at the reported time, we used the FFmpeg library.
To crop the video in space, we used the OpenCV library (in the code, we crop each image just before the pose estimation).

III. Pose estimation

Several pose estimation frameworks exist, such as MediaPipe from Google [2] or OpenPifPaf from EPFL [3]. Both achieves comparable results on our dataset but MediaPipe is a lot faster. As the dataset is big and as it take some time to process each video, we decided to go with MediaPipe. This framework runs pose estimation image by image and works only with one person. To ensure that the right person (i.e. the climber) is selected for pose estimation, we cropped our videos to remove other persons. A landmark object is created when the pose is obtained. As MediaPipe (surprisingly) does not have a function to output the landmark object to a file, we handmade a function that saves all the landmarks of a video to a json file as a dictionary. The obtained landmarks are shown in Figure 1. At this stage, we then have a json file containing the pose estimation landmarks and a photo of the associated boulder problem, this allows us to start working on the move sequence detection. A visibility number (between 0 and 1) is also returned by the pose detection, it indicates how visible is the body part. It can be interpreted as a confidence indicator.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Fig. 1: 33 pose landmarks.

A. Static extremities detection

To detect the holds, we have converted the landmark json file to a panda dataframe for simplicity of use. To get the best approximation of the position of the body's extremities (hands and feet), we defined the coordinates as the weighted average of different landmarks using their visibility as the weights:

- wrist, pinky, index and the thumb for the hands.
- foot index, heel, and ankle for the feet.

Then we implemented a method to detect if an extremity is static. The function `check_hold` allows us to store in a list the coordinates where a given extremity is static. The idea of this method is to compute the difference of an extremity's coordinates in a given frame with the following n frames. For example if n = 30, we will compute all the differences of the coordinates between the given i^{th} frame and the following 30 frames. If all these computed values are smaller than a threshold, it means that the extremity didn't move through these frames. Then the extremity is considered fixed on the i^{th} frame. This is applied for the all frame of the video, to get every frame where the extremity is fixed and its coordinates.

We used this function for all the extremities on each video to get clouds of points for each of them. Each point represents the extremity being static on a given frame (i.e. the climber either holding a hold or taking support on the wall), see Figure 2 for an example.

In this method we have two hyper-parameters, the *threshold* which represent the maximum absolute 'displacement' of an extremity to be still considered as fixed and the *number of frames* we use for the movable difference.

A threshold too big makes the borders of the clouds of points more complex to find for the clustering algorithm (because we have more points). With a too big number of frame or a too small threshold, we could not detect all the holds, especially when the climber is moving fast. A too small number of frame make slow moves to be considered as holds. Thus, after different trials of set of hyper-parameters on different boulder problems, we found that *threshold* $= 0.02$ and *number of frames* $= 30$ was a very good trade-off between the number of holds detected and the ease of clustering.

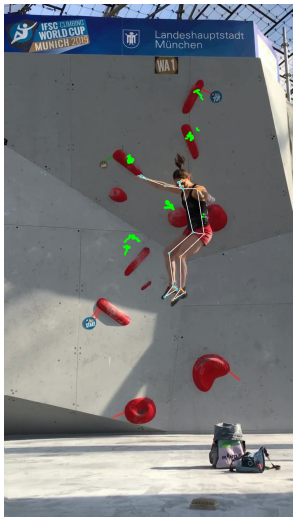


Fig. 2: Visualization of clouds of points for right hand.

Despite our stabilization, the camera can still be slightly moving, thus, the coordinate of a given point on the screenshot is not exactly the same during all the video. This is why we sometimes observe clusters of points that are not exactly on a hold but slightly shifted. A perfectly static video would fix this issue.

B. Clustering

Thanks to the function `check_hold`, we got clouds of points for each member at different places on the screenshot. The next step is to cluster those points into different groups, each one corresponding to one move.

To do so, the well-known DBSCAN [4] unsupervised clustering algorithm is used. This algorithm allows to cluster points together without specifying the desired number of clusters (as the number of move of one extremity changes between different boulder problems, or even between different move sequence of the same boulder problem, we do not have this number).

The algorithm is appropriate since it also allows to classify some points as noise. Indeed, if the previous 'static extremity detection' algorithm added only a few points at one place (because the extremity stayed still but for a short amount of time), it can be either a quick move or a noise (pure noise, or a move that is not part of the move sequence, e.g. the athlete quickly tested a hold but decided not to use it for the next move). Therefore this algorithm will be able to remove noisy points as part of the move sequence.

The algorithm has two hyper-parameters that have to be tuned: ϵ, the maximum distance between two points for one to be considered as in the neighborhood of the other, and $min\_samples$, the number of points in a neighborhood for a point to be considered as a core point. We refer to the paper [4] for a detailed explanation of those parameters.

As we have no precise accuracy measure, the optimization has to be done by hand. To do that, the clusters are computed for different values of each hyper-parameters (an example is shown in Figure 3 and 4). This is done for a lot of different boulder problems and each time the best hyper-parameters are chosen. We then aggregated the results and the chosen parameters are the following: $\epsilon = 0.03$ and $min\_samples = 20$.

To illustrate this, the following Figures 3 and 4 are the clusters computed using the algorithm. On Figure 3, $min\_samples$ is fixed to 20 and ϵ is varying between 0.01 and 0.08. In the title of each plot $n = x$ represent the number of different clusters. Each color represents a cluster and the black points are the ones considered as noise. As can be seen, the number of computed clusters is increasing when ϵ is decreasing. This number is between 4 and 6 (5 being the right choice here). For a too low ϵ ($\epsilon = 0.01$), the first move (bottom left) is classified as two different moves, and part of the two last moves are considered as noise. For a too large ϵ ($\epsilon \geq 0.06$), the two last moves are considered as one single move. $\epsilon = 0.03, 0.04, 0.05$ works perfectly here (the black points at the bottom right are indeed noisy points).

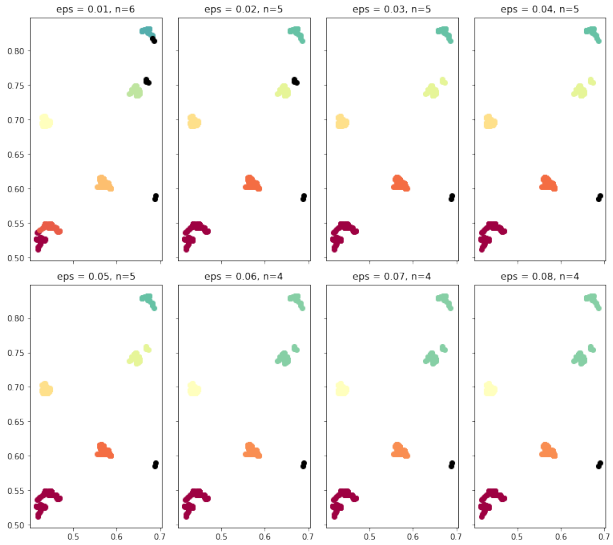


Fig. 3: Clusters computed with DBSCAN using $min\_samples = 20$ and ϵ varying between 0.01 and 0.08.

On Figure 4, ϵ is fixed to 0.03 and $min\_samples$ is varying between 1 and 200. As can be seen, for a $min\_samples$ value too low ($= 1$), the noisy points at the bottom right are not considered as noise. For a $min\_samples$ too large (≥ 75), the last hold is wrongly considered as noise. For $min\_samples$ between 5 and 50, the cluster computation is perfect. By doing this analysis for each extremity on different boulder problems, we chose the hyper-parameters stated above.

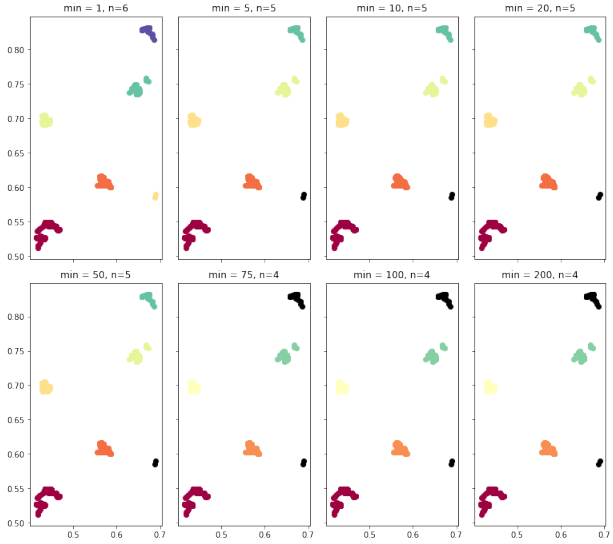


Fig. 4: Clusters computed with DBSCAN using $\epsilon = 0.03$ and $min\_samples$ varying between 1 and 200.

As we wanted to work in an unsupervised way, i.e. use the same hyper-parameters for all the bouldering problems, the algorithm cannot be perfect. Indeed, a quick move on one boulder problem can lead to the same points as noisy points

on another problem. To reach perfect accuracy, the hyper-parameters could be fine tuned on for each video, but it was not the point here. However, after testing our algorithm on a broad set of videos, the results are quite impressive and totally satisfying.

C. Visualisation

To visualize our results and checked their accuracy, we used OpenCV to draw on the screenshots of the boulders. As we can see on Figures 5 and 7 each square represents a hold or a support against the wall and is numbered to show the chronology of the sequence. This gives a clear representation of the moves sequence output by the algorithm. To improve further the readability of our results, we have used GIF, an example can be seen here.

IV. Results

A. Fail / Success move sequence comparison

Here we focus on a new boulder problem for which we have a video of a success and one of a fail. The Figure 7 shows the two associated move sequences. As can be seen, apart from the first *left_foot*, that is not detected on the fail sequence (because the foot is actually on the border of the image so the pose estimation is not consistent), the rest of the move sequence is the same until move 8 (using the numbering of the left image). Then, on the success, the climber directly reach the next hold with her right hand (move 9). On the fail, the climber try to move her left foot upper, on the black hold (move 8 and 9), then put her right hand on the same hold as her left hand (move 10[1]), then she falls. Those move sequences allowed us to see where the second climber did wrong.

Fig. 5: Success (left image) vs fail (right image) move sequences.

B. Accuracy

To check the robustness of our algorithm, a truly new dataset was needed (different camera and different boulder problems).

[1]The move 10 green square is a bit upper than the hold she actually grabs, this is because the camera is pointing more and more to the bottom after the screenshot is taken.

We thus recorded us climbing and tested our algorithms on our videos. The results were consistent with what we obtained before. This lead us to think that our process can be generalized to other datasets. Our test dataset is composed of 27 videos of 12 different boulders. We considered that there is an error in the sequence when a holds is missed, added or not in the order it should be compared to the sequence we can see on the video. For example adding a hold and having 2 holds not in the right order is considered to be 3 errors. On Figure 6 we can see that for 88% of the videos, the algorithm made two error or less. This is to our mind acceptable given that the boulders studied here are between 13 and 25 moves long. On the Figure 7, the move sequence given by the algorithm is the same as when done by hand except that the movement of the camera during the video shifts the algorithm move sequence upward and a first right foot is labeled (move 2) while the climber was still on the ground. This is not considered as an error because it is labeled before the beginning of the climb.

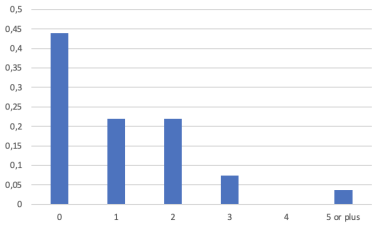


Fig. 6: Distribution of the number of errors in the move sequence given by the algorithm.

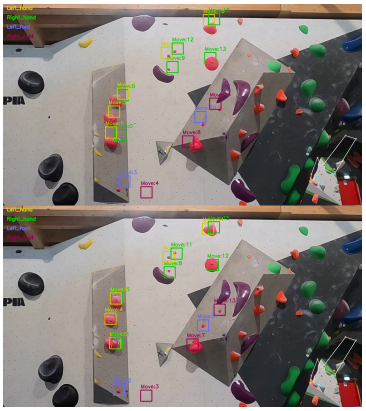


Fig. 7: Move sequence labeled using our algorithms (upper image) and by hand (lower image).

V. Conclusion

The results we obtained are quite convincing and can be generalized to other datasets. They could be used to as input for a model predicting the success rate of an athlete given a particular problem, or even while trying to predict the most successful route or move sequence. Some metrics could also be used to estimate the 'distance' between two routes.

REFERENCES

[1] Sean Csusak. Identification and classification of holds for a rock climbing wall. https://github.com/scsukas8/NeuralClimb, 2016.

[2] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. Mediapipe: A framework for building perception pipelines, 06 2019.

[3] Sven Kreiss, Lorenzo Bertoni, and Alexandre Alahi. Openpifpaf: Composite fields for semantic keypoint detection and spatio-temporal association, 2021.

[4] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.