Automation of Trampoline Competition Judging

Submitted December 2019, in partial fulfillment of
the conditions for the award of the degree **MSc Computer Science with Artificial Intelligence.**

psycg2 - 14298750

Supervised by Graham Hutton

School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature _____

Date _____ / _____ / _____

Abstract

This project was started with the intention of finding automated methods of deducing the score given to a routine in a trampolining competition. This task is traditionally performed by a large group of up to 8 qualified judges. This project aims to automate as much of this process as possible and provide a roadmap for possible future development with the goal of eventual adoption into both competitive and training environments.

Contents

Chapter 1

Introduction

The goal of this project is to automate the judging of trampoline competitions in a manner that will allow for the distinct elements of the judging process to be separated and used individually as training aids. In a competitive situation, this tool aims to be used not as a drop-in replacement for a judging panel but as an aid to judges in order to promote consistency across multiple panels. A core design principal maintained throughout the entire project is the notion of accountability; allowing the operator to easily and effectively debug any errors that occur during production use in real time.

The intended use case for this system is not one of total autonomy as user confidence in "black box" systems is generally low, especially as the data returned from the system may affect performer scores in a highly competitive environment. Instead, the system will be developed with the intent of use by a real time operator who, as previously mentioned, should be able to debug and assess the reasoning of the system's decisions without requiring extensive technical training.

1.1 Motivation

At present, the judging of trampoline competitions, under Fédération Internationale de Gymnastique (FIG) regulations, requires up to 8 qualified judges per panel, all fulfilling various roles. This system, whilst currently functional, would benefit from advancements in computer vision and machine learning.

As well as being an aid in a competitive environment, the training environment would benefit from elements of this system as well. If a performer could gain even a rough idea of what their score would be when competing a given routine, they would be able to more effectively target their work not only on the individual skills that fail them, but the other elements of their routine that cause problems, for example Time of Flight (ToF) or Horizontal Displacement (HD). These elements, especially HD, have only become largely

influential within the most recent Olympic cycle (2017 - 2020) and hence have not had the benefit of the attention of coaches within the training setting whilst more traditional elements like execution and difficulty have been the main focus of both coaches and performers. If a system can evaluate these elements without requiring the direct attention of coaches, improvement in these areas may become much easier to achieve.

1.2 Related Work

Whilst there are no systems that provide coverage of all of the elements of judging, time of flight and horizontal displacement are currently both automated in higher level competitions. The Horizontal Displacement Measurement Device (HDTS) was created by Katja Ferger and Michel Hackbarth at the Institute of Sport Science at University Gießen, Germany [3]. This method uses force sensor plates positioned under the four feet of the trampoline to determine how long a performer is in flight and where on the trampoline they land.

This project does not aim to replace this already established system in competition mainly due to the inherent inaccuracies present in performer tracking from video data but also due to the system only using video data from the long side of the trampoline and would hence be extremely unreliable in detecting side to side movement. However, this system has potential applications in the training environment by providing an approximate value for a performer's time of flight. This trade off would make sense due to the relative costs of the system: a HDTS system for one panel (two trampolines) is sold on Gymaid's website for £6960 including VAT[1]. This is in contrast to the hardware required by this system: only a camera, already present at most competitions for routine review, is required. For both systems a computer is required for operation and feedback.

[1] HDTS listing on Gymaid website, accurate as of 06/12/2019.

Chapter 2

Background and Intended Work

2.1 A Brief Introduction to Trampoline Judging

In order to understand the areas of trampoline judging that this system aims to automate, one must first have a basic grasp of their definition and how they are currently implemented, as well as the basic organisation of a trampoline competition. All of these rules are technically defined in the British Gymnastics Code of Points (BGCoP) for the 2017 - 2020 Olympic Cycle [1]. These rules shadow the FIG Trampoline Code of Points [4]. Most trampoline competitions within the UK use an inheritance model for rule setting: all rules are the same as those defined in the BGCoP unless explicitly stated otherwise. The implications of this fact will mean that the system will need to operate with several different rulesets that would ideally be configurable by the operator again with little technical training.

2.1.1 General Structure

At any competition, three routines of ten skills are performed (BGCoP [1] section 1.1). These routines are informally named "set", "voluntary" and "final" respectively. The set and voluntary routines are competed by all competitors during the qualifying round and then the eight highest scoring competitors go on to compete a final routine (BGCoP [1] section 1.3.1). At some competitions, the final round is omitted and the final placings are decided by the qualifying round.

In lower level categories, the set routine is chosen from a prescribed set of one or two routines given by the competition organisers and the voluntary and final routines can be chosen by the competitor providing they adhere to guidelines given by the competition organisers. These guidelines can encompass mandatory skills, forbidden skills and lower and upper difficulty limits. In higher level categories, the set routines are generally treated as voluntary routines in that they are free to be chosen by the performer. An emergent

behaviour among lower level performers is to perform the same routine as both their set and voluntary, this serves to simplify training regimens as only one routine needs training and remembering. This practice is termed "set repeat".

In most cases, set routines do not garner Difficulty (DD) scores as these routines are the same for all competitors. In categories where the set routine acts as a voluntary, general practice is to allow one or two skills to gain DD scores. These skills are dictated by the performer before competing by notating them on their Tariff sheet with an asterisk, hence the term "starred elements".

If a performer does not complete a routine, there are differing rules on what occurs. At most events, only the skills completed will be awarded scores which in most cases puts a competitor behind any other competitor that has completed all ten skills. At some events, an unfinished routine will get a score of 0.

2.1.2 Skills

Skills consist of three elements: somersault, twist and shape. Skills can only have one shape. Somersault rotation is divided into unit amounts of 90 degrees and twist rotation divided into unit amounts of 180 degrees. Twists are sectioned depending on which somersault they take place in. For example, if a tucked double somersault had half a twist rotation in the first somersault and another half twist in the second somersault (Half In Half Out), this would be classed as a different skill than a tucked double somersault with two half twists in only the second somersault (Back In Full Out). This is despite both skills having the same shape, amount of somersault and twist and hence identical tariff. This point is of note as repetition of skills within a routine is illegal in competition and can garner penalties ranging from exclusion of that skill's DD score to the nullification of all scores of the offending routine; effective disqualification. The exact penalty differs depending on the competition.

Shape

There are four shapes used in skills: straight, tuck, pike and straddle. The straddle shape can only be used on its own, with no somersault or twist. If a skill consists of only twist with no somersault, it must be straight. The shapes are shown in Figure 2.1

Landing

Skills can also be classified by their landings, which group them into one of four groups: feet, front, back and seat. Feet landings are by far the most common, and seat the least. Much like with twist timing, a multiple skills can be identical in all ways but this one

(a) Straight shape (b) Tuck shape (c) Pike shape

Figure 2.1: The three somersaulting positions used in trampolining.

factor. For example, a tucked double back somersault (from feet to feet) has the same somersault, twist distribution, and shape attributes as a double bounce roll (from back to back) but is a separate skill.

FIG Notation

In order to notate skills more efficiently than using their canonical names, FIG notation is used (BGCoP [1] appendix I). The system is as follows:

- The first digit describes the amount of unit somersault rotations.
- Subsequent digits describe the amount of unit twist rotation in each full somersault.
- The shape character denotes the shape of the skill, `<`, `/` or `o` for pike, straight and tuck shapes, respectively.
- In some cases, an optional landing symbol is used to denote this attribute, `f`, `b` or `s` for front, back and seat landings, respectively. Feet landings have no landing character.

This notation will be used to clear up the ambiguity present in the previous two examples. A tucked Half In Half Out is notated as `8 1 1 o` whereas a tucked Back In Full Out is notated as `8 0 2 o`. Whilst these skills have the same constituent parts, the twist distribution makes them distinct. Using the extended notation, a tucked double back somersault is notated as `8 0 0 o`, in contrast to a tucked double bounce roll which is notated as `8 0 0 o b`. Here, the landing character makes these skills distinct.

It is important to note that direction of somersault plays no part in notation or scoring as the lines of a forward or backward somersault become blurred once twists are added. For example a Front In Half Middle Back Out (`12 0 1 0`) is a triple somersault with

both entire front and back somersaults. For simpler skills, direction of somersault can be inferred from the amount of unit twists: generally an even number of half twists represent a back somersault and an even number a front somersault. This is due to the danger inherent in landing a somersault "forwards", as this is a blind landing.

This notation is important as it can convey the entire range of possible skills without having to hard-code the canonical names. It is the predominant notation that will be used in this system.

2.1.3 Difficulty

Difficulty, DD, or tariff, is an objective score given based on the complexity of the skills performed. Rather than an arbitrary mapping from skill to score, the tariff of a given skill is calculated by a function that takes into account the shape of the skill and the amount of quarter somersault rotations and half twist rotations. The function is defined as follows (BGCoP [1] section 18.1.1):

- Add 0.1 per 1/4 somersault rotation
- Add 0.1 after complete single somersault
- Add 0.1 after complete double somersault
- Add 0.2 after complete triple somersault
- Add 0.2 after complete quadruple somersault
- Add 0.1 per 1/2 twist rotation
- Add 0.1 per complete somersault in either the pike or straight shape, unless the somersault is less than a double and has twisting rotation

For example, a backwards rotating tucked double back somersault would garner $8 \times 0.1 = 0.8$ for the somersault rotations, 0.1 for a complete single somersault and 0.1 for a complete double somersault. With no extra points awarded for shape as the skill is tucked, the total score is 1.0. The rules for determining tariff are consistent across all trampoline competitions worldwide.

The sum of all the scores for all 10 skills constitutes the whole routine tariff.

2.1.4 Execution

Execution, or form, is a subjective score given based on the "neatness" of a routine (BGCoP [1] section 21.3). It is given by up to 6 judges (BGCoP [1] section 19.1). Each

judge gives a deduction of up to 0.5 from a score of 1.0 for each skill and adds each skill score to come to a routine score with a maximum value of 10.0. The median two scores are then added together to provide the resultant execution component (BGCoP [1] section 18.2.5.2).

2.1.5 Horizontal Displacement

The HD score is determined by where on the trampoline the performer lands. A deduction for each skill is taken from 1.0 and these scores are added to make a score with a maximum value of 10.0 (BGCoP [1] section 18.2.6.2). This element is handled by the HDTS system at most high level competitions but when this is not available it can be scored by 2 judges, the average of the scores of whom are taken to be the final HD score. The deduction map is given in Figure 2.2

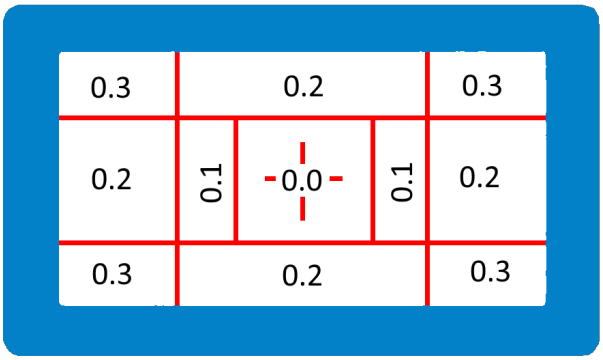


Figure 2.2: Map showing deduction given per skill dependant on where the performer lands.

2.1.6 Time of Flight

The ToF score represents the time spent in the air during the routine. This score is only present when a HDTS system is used. The points awarded correspond to the number of seconds spent during the routine measured in 1/1000 seconds and rounded down to the nearest 5/1000th.

Chapter 3

Methodology

3.1 Pipeline

3.1.1 Data batching

In order to enable threading of the workloads and to account for slow movement of the background from audience or performers behind the target performer, the video is separated into batches, the size of which is dictated by the specifics of the backplate generation. A separate backplate is generated for each batch. The default batch size is the result of the equation $B_{count} \times B_{gap} + B_{cooldown}$ where the constants are defined as follows:

- **Backplate frame count (B_{count}):** dictates the number of frames used in the backplate.
- **Backplate frame gap (B_{gap}):** dictates the number of frames between each frame used in the backplate. This is useful for having a backplate that takes into account a large time span but does not use large amounts of memory.
- **Backplate cooldown ($B_{cooldown}$):** denotes how long beyond the time waited to ensure no two frames are in any one backplate the loop waits to generate a new one.

As an example, for a video of 819 frames and constants $B_{count} = 40, B_{gap} = 4, B_{cooldown} = 20$, the batch sizes are as follows:

Batch index	Batch size (frames)
0	180
1	180
2	180
3	180
4	99

3.1.2 Image Preparation

First, the frames are read from the input video file and are cropped to remove excess data from the left and right of the bed. This results in Figure 3.1.

Figure 3.1: Frame after cropping

3.1.3 Backplate Generation

The backplate for the batch is generated by taking a frame every B_{gap} frames until the buffer contains B_{count} frames. The buffer then undergoes a median average along the 0^{th} axis. The result of this is then the backplate for the batch as seen in Figure 3.2.

The effect of this is to remove the fast moving temporal changes to each pixel generated by the performer as they jump. The resultant backplate can be subtracted from each frame to isolate these fast moving changes. As the backplate is only valid for it's batch, meaning slower moving changes from people in the background do not propagate throughout the whole video.

3.1.4 Performer Position Tracking

This sub pipeline is performed for each frame within the batch.

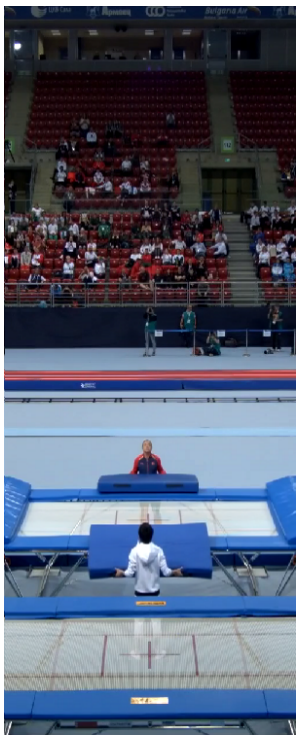


Figure 3.2: Backplate for a batch, note the performer is not visible.

Backplate Difference

The backplate is subtracted from the frame image giving Figure 3.3a. The performer is shown isolated with some noise resultant from the slower movements from within the batch.

Difference Threshold

The difference is next converted to grayscale and a threshold is taken to produce a binary image. The threshold value of 7 was taken as it has proven effective during testing but in further iterations this should be either configurable by the user or, if some fitness function can be found, simple optimisation could be implemented to ensure optimal isolation of the performer can be achieved.

Clean Threshold

The resultant boolean image is cleaned by an opening with a disk kernel of radius 5. This radius has proven effective in testing but may require tweaking if the optical hardware used features temporal noise that would have an effect of the cleanliness of the prior thresholding.

Contour Detection

Finally, the contours of the boolean image are found in order to detect the performer. In Figure 3.3d, the best case scenario is shown where only one contour is detected and hence the performer is obvious.

However, in Figure 3.3e, another frame is shown where multiple distinct contours are detected. In this case, steps are taken to make use of the other contours:

1. The contour with the largest area is selected as the root, shown in green.
2. All other contours are evaluated to determine if a point along their edge is within 40 pixels of any point along the edge of the root contour. All contours that apply are shown in blue.
3. All other contours are disregarded and are shown in red.

As can be seen in the example, this method is too conservative as it discounts a contour that clearly contains the performer. In the future, methods could be explored to effect a "chaining" method that treats all contours accepted in the first round of expansion as the root contour and adds surrounding contours until no more candidates are viable. However, the current method still proves to be effective especially when other frames contain more complete contours.

After the list of viable contours, shown in green and blue, is found, the mean average position of the contours, weighted by their area, is found. This returns the performer centroid for the current frame, shown in turquoise. This point is then temporally smoothed to reduce anomalies as a result of non-root contours being added and removed from the viable list on a frame by frame basis. The smoothed performer centroid is shown in pink.

Extremity Detection

In order to find other neighbouring valid contours, each pair of every point of every valid contour is looped over. Within this loop it is possible to determine the most distant pair of points as shown in Figure 3.4. It stands to reason that a line then drawn through these points reflects the current angle of the performer and will change smoothly with time, even when the shape of the performer changes. The angle of the performer is then calculated using the $arctan2$[1] function. When no extremities can be found for a frame, the angle is linearly interpolated.

[1] https://en.wikipedia.org/wiki/Atan2

Confidence calculation

Each frame has a position confidence and an extremities confidence. This is used later to reduce the effect of low level errors propagating through to higher level calculations. The position confidence is calculated as follows:

$$\alpha_p = \sqrt{max(\frac{\sum_{c \in C} c_{area}}{5000}, 1)}$$

The extremities confidence is calculated as follows:

$$\alpha_e = \left(1 - \frac{\left\| \frac{E_1 + E_2}{2} - \mathbf{p} \right\|}{100}\right) \alpha_p$$

Where:

- **Position confidence (α_p):** Value of the confidence in the position calculation, from 0 to 1.
- **Extremities confidence (α_e):** Value of the confidence in the extremities calculation, from 0 to 1.
- **Contours (C):** A set containing all of the contours accepted to be part of the position calculation.
- **Contour area (c_a):** The area of contour c
- **Extremities (E):** The set containing the two performer extremities as vectors, calculated for the frame
- **Position (p):** The vector representing the position of the performer.

3.1.5 Position Smoothing

To accurately index the individual skills, the position of the performer must be smoothed. The smoothed average for any frame n is calculated as:

$$\mathbf{S_n} = \frac{\gamma \mathbf{P_{n-}} + \mathbf{P_n} + \gamma \mathbf{P_{n+}}}{2\gamma + 1}$$

Where:

- **Smoothed position ($\mathbf{S_n}$):** The smoothed position for frame n.
- **Position ($\mathbf{P_n}$):** The calculated, rough position for frame n.
- **Previous valid position ($\mathbf{P_{n-}}$):** The rough position for the previous valid frame.

- **Next valid position ($\mathbf{P_{n+}}$):** The rough position for the next valid frame.
- **Smoothing factor (γ):** The factor controlling the intensity of the smoothing, a larger value results in greater smoothing

The term "valid" frame denotes a frame with an extremity confidence (α_e) of greater than 0.1. If, for example, when assessing the frame n, the frame $n+1$'s extremity confidence value was less than 0.1, frame $n+1$ would be used, and so on. This amounts to a weighted average of the current position $\mathbf{P_n}$ and the valid temporally adjacent positions, $\mathbf{P_{n-1}}$ and $\mathbf{P_{n+1}}$ where the adjacent positions are both weighted by γ. If it is impossible to find valid adjacent frames, at the start or end of the video, these vectors and their weights are omitted from the calculation.

3.1.6 Performer Velocity Deduction

The positions are differentiated using the following equation:

$$\mathbf{V_n} = \begin{cases} \mathbf{S_1} - \mathbf{S_0} & \text{if } n = 1 \\ \mathbf{S_{n_{max}}} - \mathbf{S_{n_{max}-1}} & \text{if } n = n_{max} \\ \dfrac{\mathbf{S_2} - \mathbf{S_0}}{2} & \text{if } n = 2 \\ \dfrac{\mathbf{S_{n_{max}}} - \mathbf{S_{n_{max}-2}}}{2} & \text{if } n = n_{max} - 1 \\ \dfrac{\mathbf{S_{n+2}} - \mathbf{S_{n-2}}}{4} & \text{otherwise} \end{cases}$$

Where:

- **Smoothed position ($\mathbf{S_n}$):** The smoothed position for frame n.
- **Velocity ($\mathbf{V_n}$):** The velocity vector for frame n.

This is numerical differentiation with a step size of 2 that also accounts for the start and end of the sequences by reducing the step size as appropriate.

3.1.7 Skill Indexing

With the y value of the performer, the system can detect when a jump starts and can count them. This is achieved by performing a reduction operation on the set of frames. This reduction operates on with a tuple of two "carried" variables. The first of which is a carried boolean value, C_a representing whether the performer is ascending or not. The function detailing C_a for any frame F_n is detailed as follows:

$$C_a(F_n) = \begin{cases} y(F_1) < y(F_2) & \text{if } n = 1 \\ \top & \text{if } \Delta y(F_{n-1}) < 0 \wedge \Delta y(F_{n+1}) \leq 0 \wedge \neg C_a(F_{n-1}) \wedge y(F_n) > 100 \\ \bot & \text{if } \Delta y(F_{n-1}) > 0 \wedge \Delta y(F_{n+1}) \geq 0 \wedge C_a(F_{n-1}) \wedge y(F_n) < 100 \\ C_n(F_{n-1}) & \text{otherwise} \end{cases}$$

The second is the carried current skill index, C_i. The function detailing C_a for any frame F_n is detailed as follows:

$$C_i(F_n) = \begin{cases} 0 & \text{if } n = 1 \\ C_i(F_{n-1}) + 1 & \text{if } \Delta y(F_{n-1}) < 0 \wedge \Delta y(F_{n+1}) \leq 0 \wedge \neg C_a(F_{n-1}) \wedge y(F_n) > 100 \\ C_i(F_{n-1}) & \text{otherwise} \end{cases}$$

Where:

- y **position** ($y(F)$)**:** A function that return the vertical displacement of the performer from the bed in frame F.
- y **velocity** ($\Delta y(F)$)**:** A function that return the vertical velocity of the performer in frame F.

The detection of individual jumps is fairly trivial: a new jump is started when the velocity transitions from being negative to positive (at the "bottom" of the bounce). This signal is debounced to account for noise in the velocity value and required the performer to reach a certain height before the "top" of the bounce is recognised.

Figure 3.5 shows 3 attributes of the performer:

- The blue line shows the y value of the performer relative to the trampoline surface.
- The orange line shows the y component of the velocity of the performer. It is easy to see the constant downwards acceleration shown by the straight lines when the performer is in mid air with no forces other than gravity affecting them.
- The green line shows the index of the jump counted since the start of the video, this value is multiplied by 50 for ease of reading.

3.1.8 Skill Angle Deduction

When locating the extremities of the performer, the ordering of the extremities is random. However, in order to calculate a cumulative angle for the whole skill, these extremities

must be consistent. This means that the extremity that starts at the head of the performer must stay at the head of the performer for the whole duration of the skill, and the same for the extremity at the feet.

To keep the extremities consistent, the algorithm must first determine when they are not and hence a flip is required. $\Delta\theta$ is the frame-on-frame difference in calculated angle. Normally this value is below 90°. However, when $\Delta\theta$ is closer to 180°, a flip in the extremities has likely occurred. Additionally, $\Delta\theta$ can also reach 360°as the $arctan2$ function returns values up to 360°, when this loops, $\Delta\theta \approx 360°$.

Henceforth, a decision map can be generated depicting the further processing of the data as dictated by $\Delta\theta$, shown in Figure 3.6. The segments are explained as follows:

- The green segment dictates acceptable somersault rotation of the performer. When $\Delta\theta$ lands within this zone, it is added to the cumulative skill angle.
- The red segment dictates a possible flip of the extremities in relation to the last frame. If this is detected, the extremities are flipped back and the angle re-evaluated.
- The blue segment indicates the $arctan2$ function wrapping around. When $\Delta\theta$ lands within this zone, nothing happens.

As the performer can undergo both clockwise and anticlockwise rotation, different decision maps must be used. Figure 3.6a is used for clockwise rotation, where $arctan2 > 0$ and Figure 3.6b for anticlockwise rotation, where $arctan2 < 0$.

3.1.9 Pose Detection

A neural network is used to detect the positions of the joints of the performer. The specifics of the network used are discussed in Section 3.2.1.

Data Preparation

Initially, the network was evaluated using the focused image of the performer as input, as in Figure 3.7a. However, whilst the poses that were detected were accurate, the performer was rarely recognised. This proved to be a product of the fact that the network was trained only on natural photos of humans, in which they are mostly standing or sitting. This mean that when the performer was inverted during a somersault, the input data was outside the range of the training data.

To combat this, the input data is first rotated to show the performer with their feet down, as in Figure 3.7b. This yielded far improved results as shown in Figure 3.7c. Frames are also padded in order to avoid issues when the performer is close to the edge of the frame.

Detection

As the network cannot tell which human to detect, it returns all of the poses it can see, as in Figure 3.7c. In order to ensure only the relevant pose is selected, each candidate pose is scored as such:

$$score(P) = \left\| \frac{\frac{\mathbf{P_{LHip}} + \mathbf{P_{RHip}}}{2} + \mathbf{P_{Neck}}}{2} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \right\|$$

Where:

- **Pose (P):** The candidate pose to be evaluated. All coordinates of the pose are normalised from 0 to 1 in the coordinate space of the input data, e.g. Figure 3.7b. Elements of the pose relate to the positions of the notated body part. For example, $\mathbf{P_{Neck}}$ is the vector containing the coordinates of the neck of pose P.

This amounts to scoring each pose with the distance from it's geometric centre to the input data frame, where the performer is located. The lowest scoring pose with a score of less than 0.2 is accepted. The cut off limit of 0.2 is used as any pose within that range will likely be the target pose as all other poses that would be within that limit will be occluded by the performer.

3.1.10 Shape Detection

After valid candidate poses are found, they are analysed to find the shape they represent. This requires the use of two data points:

- The knee angle: calculated as the angle between the hip, the knee and the ankle.
- the hip angle: calculated as the angle between the knee, the hip and the neck.

For each angle, the average of both sides of the body is used or, if only a complete angle can be detected on one side of the body, that angle is used. For each skill, the maximum angles reached are used to evaluate the shape. To do this, the angles for each skill are classified by being either greater than or less than 20°. If any angle is less than 20°, it is considered straight, otherwise it is considered bent.

The following table shows the shape assigned to each combination of straight and bent angles within a skill:

	Bent knee	Straight knee
Bent hip	Tucked	Piked
Straight hip	Straight	Straight

For visualisations of each shape, see Figure 2.1.

3.2 Implementation

The system was implemented in Python as it offered the most flexibility in its image manipulation libraries. For image manipulation, the OpenCV[2] library was used as it offered a simple python API with a backend written in C++ that enabled faster compute speeds.

As the workload by nature is split into batches, it was simple to use Python's native threading library to multithread the process. However, as Python natively a single threaded language, the overheads generated by threading resulted in a slower execution time in CPU heavy parts of the code. For sections that involved disk IO however, multithreading allowed for a significant performance improvement. This was leveraged in the first sections by reading from the disk to populate on batch whilst preceding batches begun their image processing sections. As well as this, when writing the individual skills to the disk after their identification, threading was used so as to not block the CPU heavy computations.

In future iterations, the python multiprocessing library[3] could be used as this achieves truer parallelism by using separate processes.

3.2.1 Pose detection

Initially, the neural network from the OpenPose algorithm[2] by the CMU Perceptual Computing Lab was used. However, the speed of this approach was too slow. The approach used in the final implementation was tf-pose-estimation by ildoonet[4]. This approach used TensorFlow[5] which has a versatile backend that can leverage a range of hardware for performance increases on tailored machines. This is a TensorFlow implementation of the OpenPose algorithm that allowed for a choice in the neural network used. The thin MobileNet[5] network was chosen as it resulted in faster performance due to its intended use on mobile devices. In future iterations, the original CMU network could be used in place to result in better pose detection accuracy.
The main bottleneck of the pose detection was the large difference between the data the net was trained on and the input data used within the system. When twisting, large parts of the body is occluded from view, which results in a more challenging pose for the network to resolve. In future iterations a custom trained network would be essential for more reliable results.

[2] https://opencv.org/
[3] https://docs.python.org/3/library/multiprocessing.html
[4] https://github.com/ildoonet/tf-pose-estimation
[5] https://www.tensorflow.org/

3.2.2 API

The scorer algorithm is accessed from the class: `TrampScorer`. The constructor takes the path to the root of the environment folder as a string. Within the environment folder there must be a single video file called `source`. The algorithm accepts any video extension. After construction, the `set_constants` method is called which provides various constants to the program:

- `left_cutoff:` The distance from the left side of the original frame to the leftmost edge of the cropped image as generated in Section 3.1.2. Set by the user in the default CLI.
- `right_cutoff:` The distance from the left side of the original frame to the rightmost edge of the cropped image as generated in Section 3.1.2. Set by the user in the default CLI.
- `bed_y :` The distance between the top of the frame and the level of the bed in the frame. Used for skill indexing as explained in Section 3.1.7. Set by the user in the default CLI.
- `backplate_frame_count:` The number of frames used for the backplate of each batch. Referred to as B_{count} in Section 3.1.1. Set in the default CLI to 40.
- `backplate_frame_gap:` The distance between each frame used in the backplate. Referred to as B_{gap} in Section 3.1.1. Set in the default CLI to 4.
- `backplate_frame_cooldown:` The number of frames allowed to pass after the final frame used for the backplate of each batch. Referred to as $B_{cooldown}$ in Section 3.1.1. Set in the default CLI to 20.
- **focus_width:** The width of the cropped image used as input data to the pose detection network. Set to 300 in the default CLI.
- **focus_height:** The height of the cropped image used as input data to the pose detection network. Set to 300 in the default CLI.
- **position_smoothing_factor:** The constant used in the position smoothing algorithm. Referred to as γ in Section 3.1.5. Set to 0.5 in the default CLI.

After the constants are set, the allocate_memory function is called which generates the underlying data structures and initialises the TensorFlow estimator. When this has completed, the start function starts the main body of the computation. Upon completion, the environment folder is populated with videos of the separated skills and a JSON file containing, for each skill:

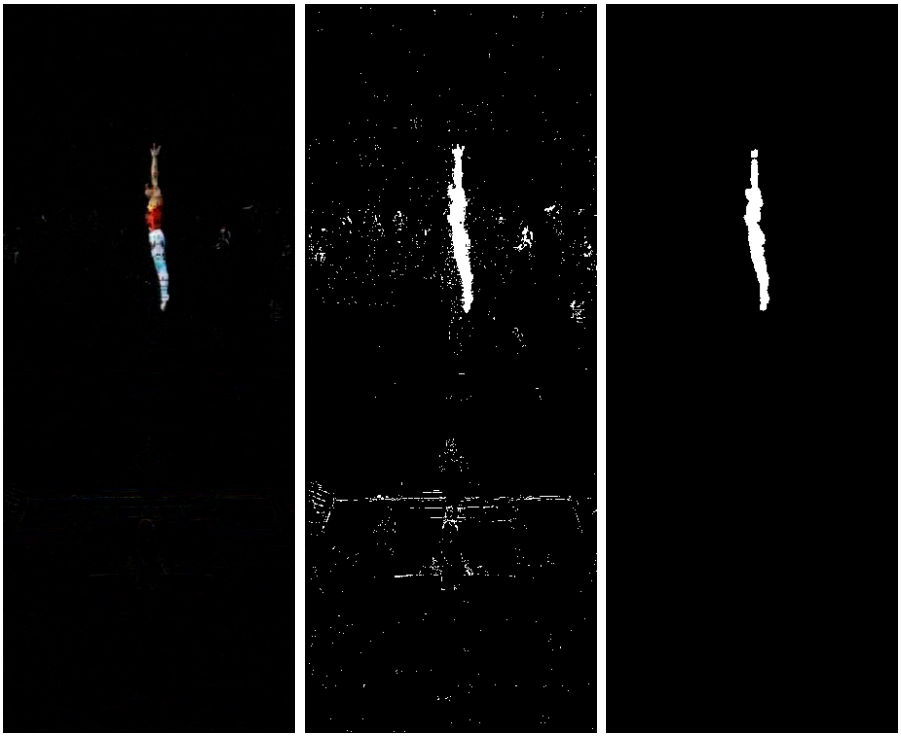
- The estimation of the skill shape
- The estimation of the somersault rotation angle
- The series of vertical displacement values throughout the skill

The user manual for the prediction script can be found in Appendix E.1

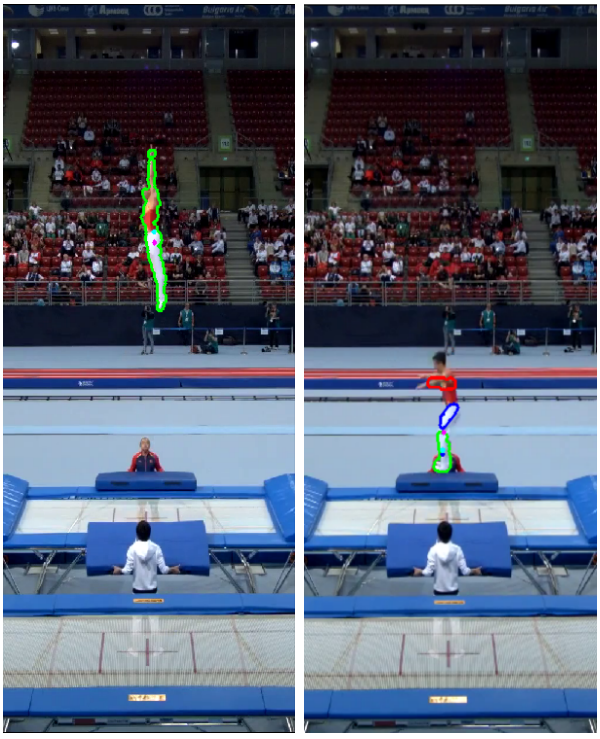
3.3 User Interface

The user interface was written using the Nuxt meta-framework[6]. It allows for the displaying of the output of the scorer program. It also allows for the user to correct the output and find the DD and ToF scores. The interface displays the segmented skills and allows for the user to play each skill individually or together in series. It also allows for the exporting of the user corrected data to put back into the environment folder for later review. The user can also mark the arm set in order to view the the details from the routine including the FIG notation for each skill. Figure 3.8 shows a screenshot of the user interface in the browser populated with test data. The user manual for the user interface can be found in Appendix E.2.

[6] https://nuxtjs.org/



(a) Backplate difference (b) Thresholded difference (c) Clean threshold



(d) Final segmentation (e) Segmentation with multiple contours

Figure 3.3: The steps of the performer location pipeline

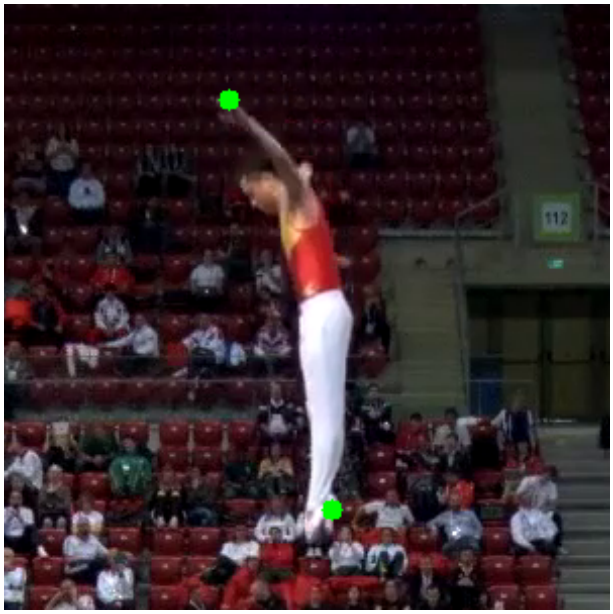


Figure 3.4: Focused frame of performer, with extremities highlighted in green.

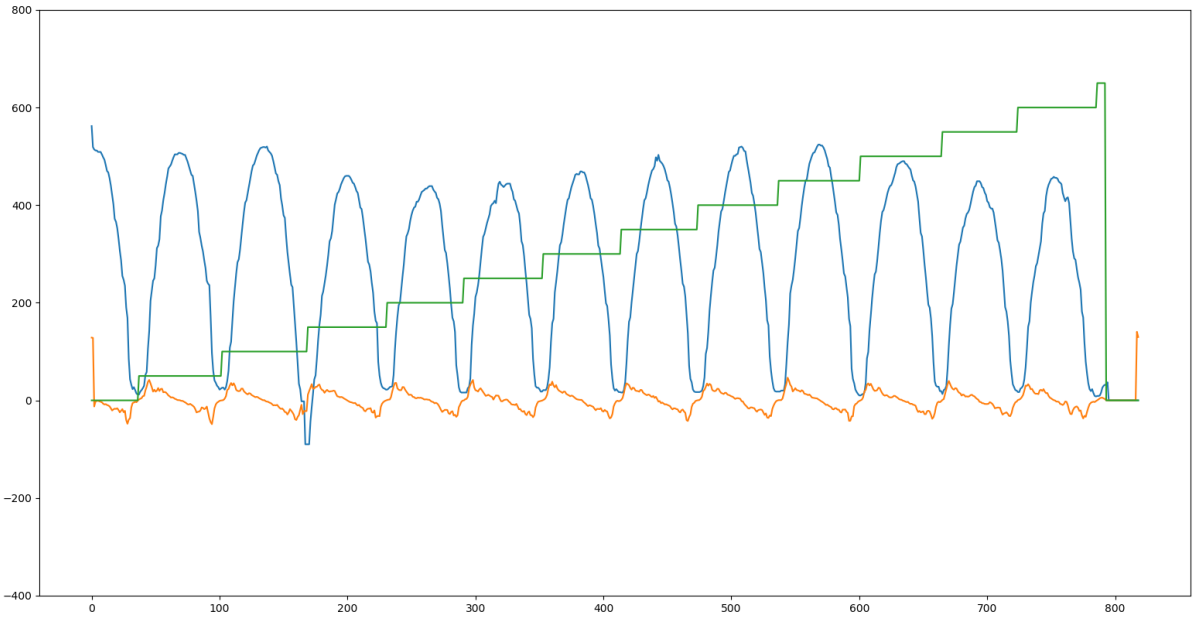
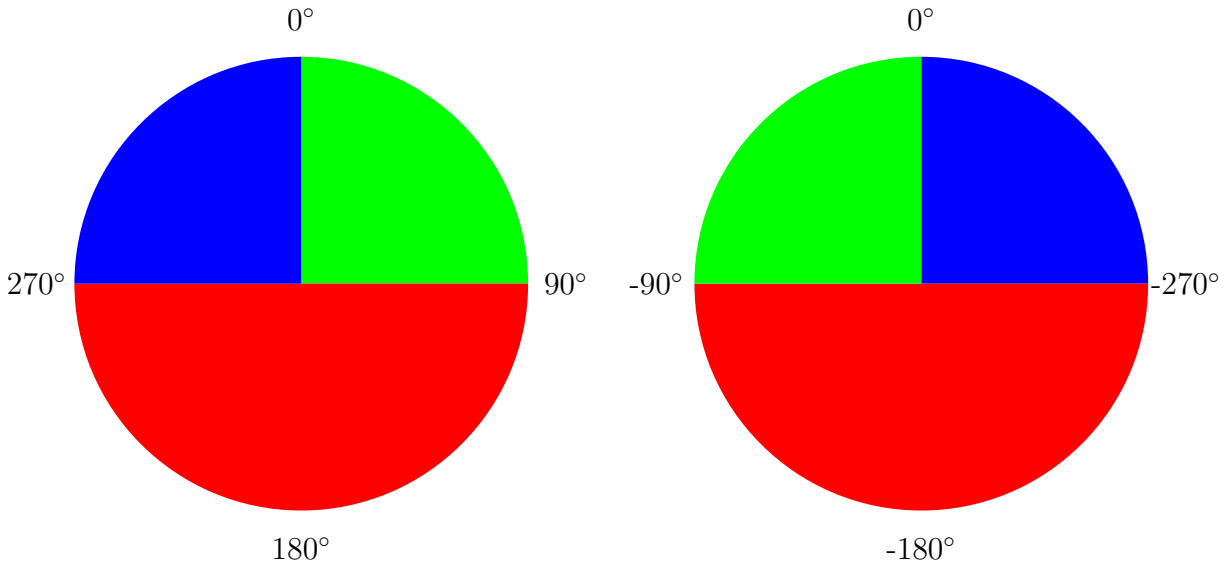


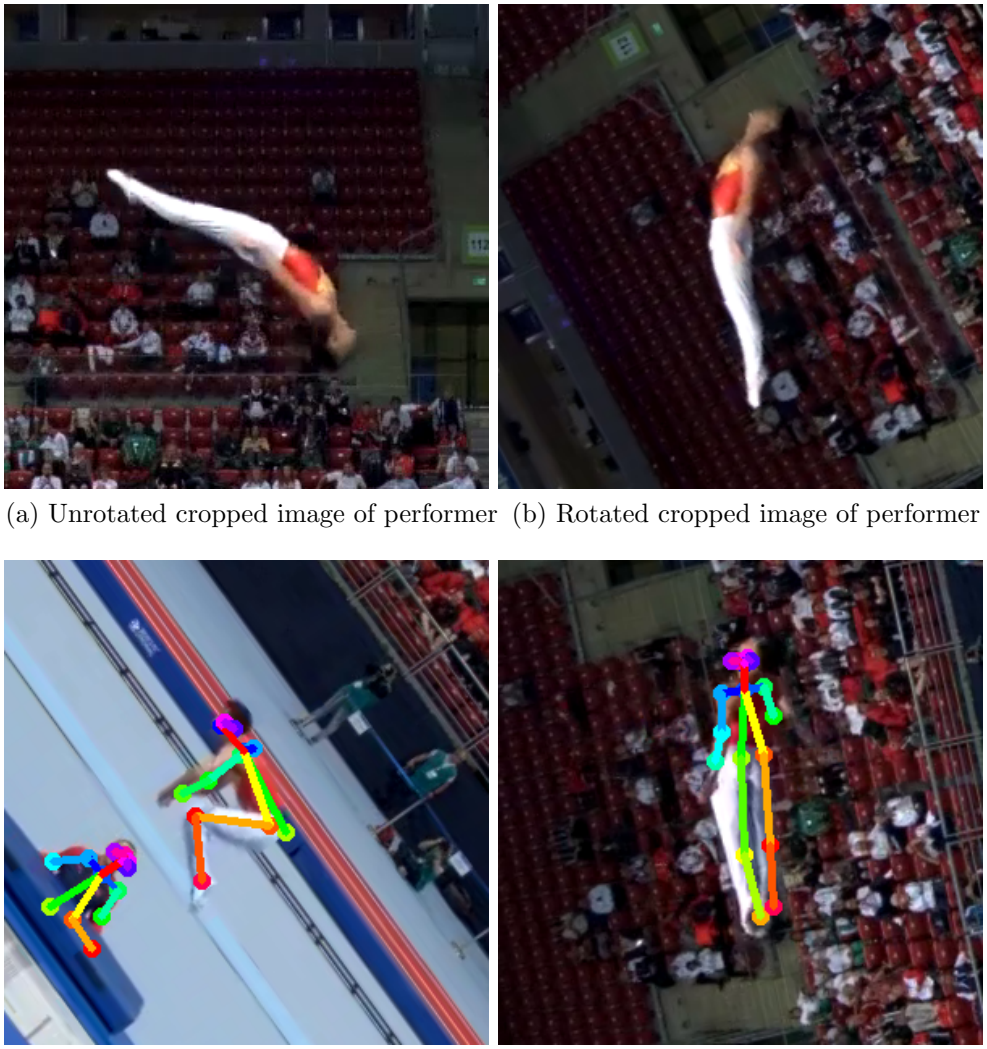
Figure 3.5: Graph showing various attributes of the performer as they jump.



(a) Decision map for clockwise rotation

(b) Decision map for anticlockwise rotation

Figure 3.6: Performer angle normalisation maps for clockwise and anticlockwise rotation



(a) Unrotated cropped image of performer (b) Rotated cropped image of performer

(c) Rotated cropped image of the performer with all detected skeletons overlaid (d) Rotated cropped image of the performer with selected skeleton overlaid

Figure 3.7: Input data given to the pose detection network before and after rotation and output data from the pose detection network.

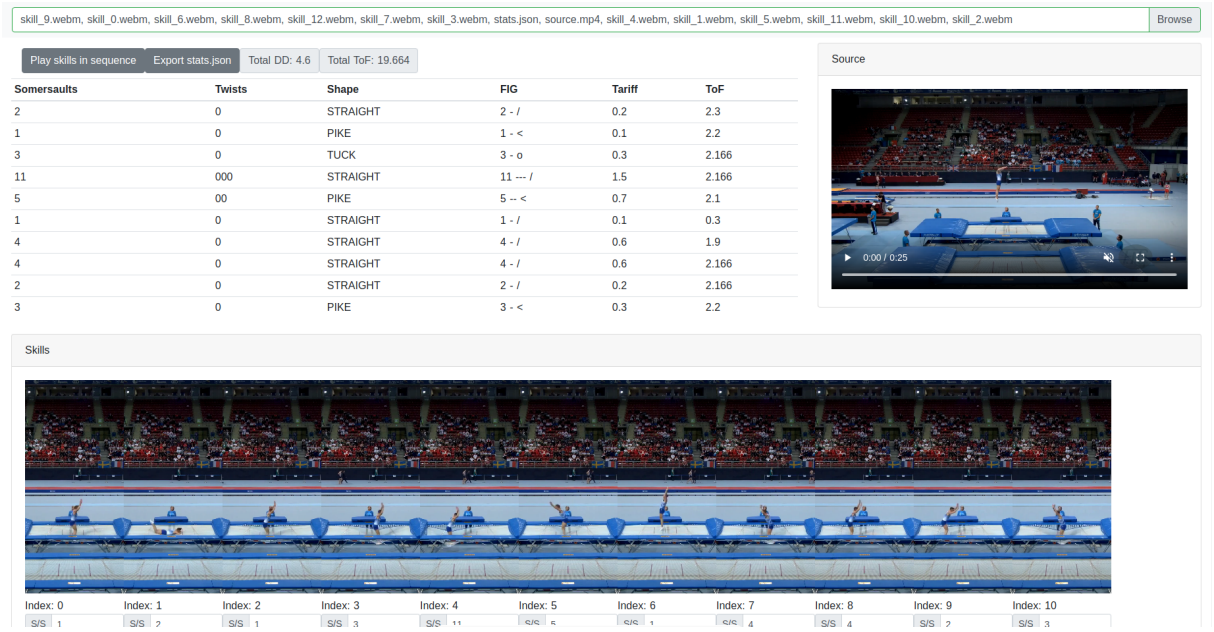


Figure 3.8: Screenshot of the user interface populated with test data

Chapter 4

Further Development

This release delivers a foundational set of capabilities for this judging and training aid. Enabling performers to identify weak skills in their routines and measure performance improvements over time. A product road map for further development is envisaged to include additional features and portability improvements, some of which are described below.

4.1 Twist Deduction

Resolving the twist angle of each skill proved a challenge as the pose estimation network was unable to provide coarse enough results to provide an accurate estimation. Whereas resolving the shape of each skill required only a single frame of results per skill, resolving the twist angle would require at least 3 frames per twist. This would provide enough data to not only deduce that a twist has occurred, but also the direction of the twist.

Assuming perfect pose estimation, where reliable data could be obtained for evrey frame, twist angle could be resolved by counting the times the performer presented face-on and in profile to the camera. For example, in a skill with a whole twist, or a twist angle of 360°, the presentation of the performer would occur in the following sequence:

1. Profile
2. Face-on
3. Profile
4. Face-on
5. Profile

Analysing these sequences of performer presentations would allow the system to resolve the complete twist angle for each skill. This data point, combined with the somersault

and shape estimations, would provide all the data necessary to compute the DD score for a routine.

4.2 Arm Set Detection

The start of each routine is usually denoted by an arm set. This skill, whilst not part of the routine, is distinct from an ordinary bounce due to the fact that the performer finishes with their arms above their head. This puts the performer in the ideal posture for starting a somersault and hence the routine. This could be detected, assuming perfect pose estimation, by measuring the angle of the performer's shoulders in the same way that both hip and knee angles are detected already. With this part of the system, the reliance on human intervention could be eliminated completely. At present, the previously mentioned system that resolves ToF and HD[3] requires a person sat at a computer to indicate to the system when an arm set has occurred and hence when the routine has commenced.

4.3 Execution Evaluation

The most novel promise of the full effectiveness of this system would be to provide an execution score for a performer's routine. This would help remove the most subjective parts of competition judging and would be a great stride in the technology used in competitive trampolining.

To do this, vast amounts of data would be required. Specifically, training data would have to be collected with the execution score of each skill annotated. With this data, a convolutional LSTM[7] would likely be the most effective tool to estimate this value in the final system.

For this approach, providing an effective pose estimation network could be obtained, the competitor's pose could be used as the input data and the execution score of each skill would be used as the target output. Further to this, transfer learning[6] could be used to provide a more complete and possibly more accurate solution by pre-training a thinner network to correlate pose positions to execution scores and concatenating this network with the pose estimation network for further training. This experimental technique could result in a higher accuracy.

4.4 Portability Improvements

With the advancements currently being made in the high powered web field, technologies such as WebAssembly[1], WebGL[2] and in the future, WebGPU[3], will allow for near native speeds of highly specialised code to be run in the browser. This fact is already emerging in the form of the browser based TensorFlow implementation tf.js[4] that uses WebGL as a backend. Additionally, tf.js has an experimental backend in WebGPU[5] that will allow neural networks running in the browser to leverage the massive parallel compute powers afforded by GPUs in a large variety of mobile and desktop devices.

Possibly the largest tool accessible as a result of these projects is the ability to reliably run the same code on any edge node platform, without the need for any native code. If this path were followed, the current codebase would have to be rewritten in a low level compiled language such as Rust[6]. The main advantage of Rust is the increased trust one can put in such a strongly typed language whilst still leveraging the speed of a compiled language which results in far greater reliability and safety than languages like C or C++. Additionally, Rust can be compiled to WebAssembly which means it can be run within the browser. All these technologies used in concert would allow for the realisation of the entire system to be run on the end user's machine. This would reduce operating costs and reduce the overall complexity, hence reducing the chance of failure.

4.5 Remote Compute

Whilst moving all of the compute efforts to the edge would result in lower running costs, upfront costs would be large as each panel (set of 2 trampolines, of which there can be up to 8 per competition) would require a machine capable of accommodating the inescapably intense compute requirements of this system. A better solution would be to use cloud based compute power to centralise these requirements whilst edge nodes communicate and provide input in real time.

The most reasonable approach would be again to rewrite the system in a low level language in order to gain performance and safety gains over Python. The TensorFlow and OpenCV libraries also have bindings for both Rust and C++ so the conversion would likely be little more than semantic with the addition of memory management.

[1] https://webassembly.org/
[2] https://www.khronos.org/webgl/
[3] https://gpuweb.github.io/gpuweb/
[4] https://www.tensorflow.org/js
[5] https://github.com/tensorflow/tfjs/tree/master/tfjs-backend-webgpu
[6] https://www.rust-lang.org/

Chapter 5

Evaluation

Due to the limited data available, the system was tested on 3 input videos. There are 3 areas of evaluation:

- Skill separation
- Somersault rotation detection
- Skill shape detection

The test videos are of 3 performers at the 2017 Trampoline Worlds in Sofia:

1. DONG Dong (CHN)
2. MARTIN Jorge (ESP)
3. HANCHAROU Uladzislau (BLR)

5.1 Skill Separation

This test will measure how often the system accurately detects the beginning of a new skill. Table 5.1 records the evaluation details of each test video.

Test video	Frame count	Total skill beginnings	True positive detections	False positive detections	False negative detections
1	819	14	14	0	0
2	711	12	12	0	0
3	767	11	11	1	0
Total	2297	37	37	1	0

Table 5.1: Details of skill start detections

This is an encouraging result as it shows that over 2297 frames of video containing 37 skill beginnings, all 37 were positively detected, with only 1 false detection. This results in a sensitivity rate of 100% and a precision rate of 97%.

5.2 Skill Angle Detection

This test compares the evaluated somersault angle with the genuine somersault angle for each skill of each evaluation routine. Note a difference of up to ±45° is acceptable as the angles are rounded to the nearest quarter-somersault. The complete table of data from this test is in Appendix C.1

This test depicts poorer performance than the previous one. Per skill, there is an average error of 136°. Of the 30 skills within the 3 routines, the algorithm guessed 11 correctly. It is noteworthy however that for test routine 1, 70% of skills has their rotations guessed correctly, in contrast to test routines 2 and 3, of which only 20% of the skills were correctly assessed. This may be, primary, due to the colour of the performer's leotards. In video 1 the leotard is more vibrant and hence stands out more from the background whereas in the other videos, the leotards blend with the background. It seems that this did not provide too much difficulty in location the position of the performers but did not allow for the more fine-detailed image processing required to assess the somersault rotation angles.

Figure 5.1 is a plot of actual somersault rotation angles against calculated rotation angles. Test videos 1, 2 and 3 are shown in red, green and blue respectively. The closer a point is to the black line on the graph, the more accurate the prediction and points that lie within the boundaries of the grey lines are considered acceptable. As can be seen, test video 1 is more accurate than the rest as many of it's points lie within the acceptable bounds.

5.3 Skill Shape Prediction

This test consists of comparisons between the predicted shape and the actual shape of each test routine. The full data for this test can be found in Appendix C.2. Table 5.2 is a confusion matrix detailing the class predictions.

		Actual shapes			
		Straight	Pike	Tuck	Total
Predicted shapes	Straight	9	2	1	12
	Pike	3	5	1	9
	Tuck	1	4	4	9
	Total	13	11	6	

Table 5.2: Confusion matrix for skill shape class predictions

Table 5.3 shows the precision and recall of all shapes across all 3 test videos. It shows that whilst the straight shape is easier to infer, tucks and pikes are harder to differentiate.

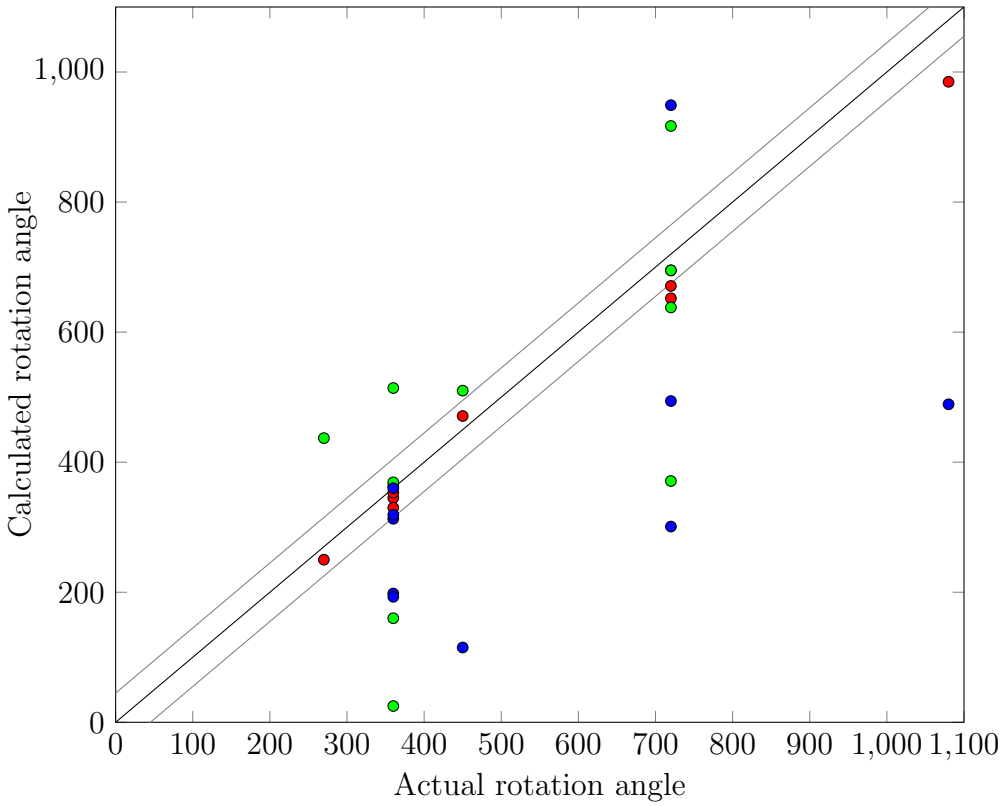


Figure 5.1: Graph of skill angle predictions, the grey lines depict the acceptable bounds

Shape	Precision	Recall
Straight	75%	69%
Pike	56%	45%
Tuck	44%	67%

Table 5.3: Precision and recall data for all shapes

Chapter 6

Reflection

6.1 Project Management

This was a very large undertaking, with many novel and complex facets. It would be appropriate to say the complete system was too large a goal for a single year dissertation project. Whilst some promising results were obtained in the areas of image and signal processing, sections that relied on artificial intelligence lacked the time and resources necessary to gain an effective result. Using pre-trained networks did not suffice for the niche and specific use cases presented within the system.

Too much time was devoted to the fine tuning of the system with the limited training data available, resulting in poor generalisation to other videos. Additionally, whilst a user interface was prepared, thi provided mainly a view of the output data, rather than a user friendly front end with which to operate the system.

With more time and data, a complete system would be possible. However, these factors were not effectively assessed when choosing the topic of the project. Due to the overwhelming scope, time was not effectively partitioned and hence too much time was spent on refining the operation of the initial, lower level features of the system and not enough time was given to ensuring a more complete and effective program. This learning merits the potential future use of a product road map approach would allow manageable releases to be constructed providing ever increasing capabilities and provide for improved planning and estimation.

6.2 Achievements

This system does a remarkable job at detecting the position of the performer within the view of the camera and can segment the individual skills with reasonable accuracy.

Additionally, the user interface is remarkably intuitive and allows the user to easily refine the output of the prediction script.

6.3 Shortcomings

A large difficulty during the development of the system was a lack of reliable training data. Only a select few videos from the 2017 Trampoline Worlds in Sofia were suitable for use during development. As well as this, whilst data on the overall execution score was available for these videos, marks for each individual skill were not present. This rendered the execution evaluation part of the system impossible to develop as these marks would be used as the output of the network trained to score this part.

Additionally, the pose detection network was too general for this specific use case and hence did not provide reliable enough data for more accurate skill shape deduction.

If this system were to be used in a competitive environment, custom trained networks would be essential. This would be a huge task as training data would have to be hand annotated, the networks would require at least pose annotation and have skill execution scores provided, which would require a set of trained judges to provide an unbiased score.

6.4 Summary

Whilst some effective strides were undoubtedly made, this project fell foul of being too large in scope and failure to accurately assess the availability of existing resources. With more time, there is no doubt that a system that lives up to it's initial promise would be possible, and the final deliverables of this project would provide a successful foundation to that goal.

Bibliography

[1] British Gymnastics. *2017 – 2020 Code of Points Trampoline, Tumbling & Double Mini Trampoline*. Link.

[2] Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S., and Sheikh, Y. A. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[3] Ferger, K., and Hackbarth, M. New way of determining horizontal displacement in competitive trampolining. *Science of Gymnastics Journal 9* (01 2017), 303–310.

[4] Fédération Internationale de Gymnastique. *Fédération Internationale de Gymnastique Trampoline Gymnastics 2017 – 2020 CODE OF POINTS*. Link.

[5] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[6] Martin, S. What is transfer learning? Link.

[7] Olah, C. Understanding LSTM networks. Link.

Appendix A

Acronyms

Acronyms

BGCoP British Gymnastics Code of Points. 3

DD Difficulty. 4, 6, 19, 26

FIG Fédération Internationale de Gymnastique, the international governing body of gymnastics. 1, 3

HD Horizontal Displacement. 1, 7, 26

HDTS Horizontal Displacement Measurement Device. 2, 7

ToF Time of Flight. 1, 7, 19, 26

Appendix B

Glossary

Glossary

FIG notation A simple and standard way of notating skills in order to avoid writing the full English names. See Section 2.1.2. 5, 19

Tariff sheet A document provided to the judging panel by competitors before they compete dictating the routines they aim to perform, as well as any special notation required by the competition rules.. 4

Appendix C

Evaluation Data

C.1 Somersault Angle Predictions

Test routine	Skill index	Actual angle	Computed angle	Error	Acceptable?
1	1	270°	250°	20°	Yes
	2	450°	471°	21°	Yes
	3	720°	671°	49°	No
	4	720°	652°	68°	No
	5	360°	345°	15°	Yes
	6	360°	363°	3°	Yes
	7	360°	330°	30°	Yes
	8	360°	353°	7°	Yes
	9	720°	695°	25°	Yes
	10	1080°	985°	95°	No
2	1	270°	437°	167°	No
	2	450°	510°	60°	No
	3	360°	25°	335°	No
	4	360°	369°	9°	Yes
	5	360°	514°	154°	No
	6	360°	160°	200°	No
	7	720°	695°	25°	Yes
	8	720°	371°	349°	No
	9	720°	917°	197°	No
	10	720°	638°	82°	No
3	1	360°	198°	162°	No
	2	450°	115°	335°	No
	3	360°	313°	47°	No
	4	720°	949°	229°	No

Table C.1 – continued from previous page

Test routine	Skill index	Actual angle	Computed angle	Error	Acceptable?
3	5	720°	494°	226°	No
	6	360°	319°	41°	Yes
	7	360°	360°	0°	Yes
	8	360°	193°	167°	No
	9	720°	301°	419°	No
	10	1080°	489°	591°	No

C.2 Shape Predictions

Test routine	Skill index	Actual shape	Predicted shape
1	1	Straight	Straight
	2	Tuck	Tuck
	3	Tuck	Tuck
	4	Pike	Pike
	5	Pike	Tuck
	6	Straight	Straight
	7	Straight	Straight
	8	Tuck	Tuck
	9	Pike	Pike
	10	Pike	Tuck
2	1	Straight	Straight
	2	Straight	Pike
	3	Straight	Straight
	4	Straight	Straight
	5	Pike	Pike
	6	Straight	Straight
	7	Straight	Tuck
	8	Pike	Pike
	9	Pike	Tuck
	10	Straight	Pike
3	1	Straight	Pike
	2	Pike	Tuck
	3	Tuck	Tuck
	4	Tuck	Straight

Table C.2 – continued from previous page

Test routine	Skill index	Actual shape	Predicted shape
3	5	Tuck	Pike
	6	Pike	Straight
	7	Straight	Straight
	8	Straight	Straight
	9	Pike	Pike
	10	Pike	Straight

Appendix D

Attached Files

The following directory tree shows the folder structure of the attached files. Note individual files are not shown.

```
/
├── eval_src ...  The source code of the evaluation algorithm
├── output_envs ...  Environment folders for the 3 test videos
│                    before the data has been corrected by the UI
│   ├── dong_dong
│   ├── jorge_martin
│   └── uladzislau_hancharou
├── corrected_envs ...  Environment folders for the 3 test videos
│                       after the data has been corrected by the UI
│   ├── dong_dong
│   ├── jorge_martin
│   └── uladzislau_hancharou
├── ui_src ...  Source code of the user interface
│   ├── layouts
│   └── pages
│       └── index.vue ...  Functional part of the source code of the user
│                          interface
└── ui_dist ...  Pre-built static files ready for hosting on an
                 arbitrary server
```

Appendix E

User Manual

E.1 Prediction Script

In order to run the script, the following dependencies are required:

- Python 3
- python-opencv
- ffmpeg
- numpy
- tf-pose-estimation[1] installed as a package

To run the predictor, run the command from the `eval_src` folder: `python main.py [path to environment root] [left_cutoff] [right_cutoff] [bed_y]`. The arguments are discussed further in Section 3.2.2. However, in order to evaluate the videos included in the sample root environments, the following arguments can be used:

- `left_cutoff`: 700
- `right_cutoff`: 1200
- `bed_y`: 730

E.2 User Interface

In order to run the user interface from source, the following dependencies are required:

- node.js[2]

[1] https://github.com/ildoonet/tf-pose-estimation
[2] https://nodejs.org/

- npm[3]

Then follow the steps:

1. Navigate to the package root folder (`ui_src` in the included files)
2. Run `npm i`
3. To run statically:
   (a) Run `npm run build`
   (b) Run `npm run start`
   (c) Then to run subsequently run `npm run start` again
4. To run dynamically (one time):
   (a) Run `npm run dev`

Alternatively, the UI is hosted statically at https://goodingc.github.io/tramp-scorer-ui/.

To operate the UI, simply open the environment folder in the file dialogue. Note the whole folder must be uploaded such that all the contents are included. Environment folders include all folders contained within the `output_envs` and `corrected_envs` folders.

To play all skills in sequence, press the "Play skills in sequence" button. In order to play an individual skill, click the video of the desired skill.

In order to calculate routine values:

1. Designate the arm set skill with the "Set as Armset" button
2. Set the somersault and shape values for each skill
3. Set the twist values for each skill where each decimal represents one complete somersault. For example, a double somersault with a half twist in each somersault will have a twist value of 11. For more information, see Section 2.1.2
4. In the event of a false positive detection, skip any skills necessary with the "Skip skill" button
5. The full routine information will be present on screen
6. If desired, click the "Export stats.json" button to export the data and replace the old `stats.json` file in the environment folder.

The environment folders within the `output_envs` folder are uncorrected, those within the `corrected_envs` have been corrected.

[3] https://www.npmjs.com/