TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

IAPM02/15
Marjana Voronina 163582

AUTOMATED CAMERA MOTION CONTROL FOR RHYTHMIC GYMNASTICS USING DEEP LEARNING

Master's Thesis

Supervisor: Kristina Vassiljeva
Associate Professor

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

IAPM02/15
Marjana Voronina 163582

AUTOMATISEERITUD KAAMERA JUHTIMINE ILUVÕIMLEMISE JAOKS KASUTADES SÜVAÕPET

Magistritöö

Juhendaja: Kristina Vassiljeva
Associate Professor

Tallinn 2019

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, the literature and the work of others have been referenced. This thesis has not been presented for examination anywhere else.

Author: Marjana Voronina

2019-05-07

Abstract

The purpose of this thesis is to explore if end-to-end deep neural networks can be used to operate a camera motion control system in order to record or broadcast rhythmic gymnastics events. Specifically, presented research aims to determine if an end-to-end neural network can produce an eye-pleasing output by tracking an individual rhythmic gymnast performing on a standard competition area.

To answer this question, deep neural networks of different types and structures were created, trained and compared to each other as well as to the performance of a human operator.

Results showed that end-to-end deep neural networks can be successfully used to operate a camera motion control system during rhythmic gymnastics events. While a pure Convolutional Neural Network (CNN) solution solves the problem, a CNN+Long Short-Term Memory (LSTM) network outperforms it.

This thesis is written in English and is 56 pages long, including 7 chapters, 29 figures, and 1 table.

Annotatsioon

Käesoleva lõputöö eesmärgiks on uurida, kas süvanärvivõrgud on sobiv vahend filmimissüsteemi juhtimiseks eesmärgiga filmida ja kajastada iluvõimlemisvõistlusi. Uurimistöö peab andma vastuse, kas süvanärvivõrk on võimeline genereerida väljundit, mille abil saaks jälgida standardsel võistlusväljakul esinevat individuaalvõimlejat.

Küsimusele vastamiseks luuakse ja treenitakse erinevat tüüpi ja erineva struktuuriga süvanärvivõrke ning võrreldakse nende toimimist omavahel ning ka videooperaatoriga.

Tulemused näitavad, et konvolutsiooniline närvivõrk (CNN) lahendab probleemi, kuid CNN+*Long Short-Term Memory* (LSTM) tüüpi närvivõrk toimib veelgi paremini.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 56 leheküljel, 7 peatükki, 29 joonist, 1 tabelit.

List of abbreviations and terms

CNN Convolutional Neural Network

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MSE Mean Squared Error

ORB Oriented FAST and Rotated BRIEF

PTZ Pan-Tilt-Zoom

RNN Recurrent Neural Network

SIFT Scale-Invariant Feature Transform

SURF Speeded-Up Robust Features

Table of Contents

List of Figures

List of Tables

1 Introduction

Many local rhythmic gymnastics competitions are not being recorded or broadcast because of high expenses required to cover camera operator's work for 10+ hours. In other words, using human labor is not cost-effective for local sport events. This problem could be solved by a fully automated system. Pure object tracking does not result in aesthetic shots with good positioning and is inherently not predictive or proactive. Neural network-based solution can reduce costs, making the process fully automated and therefore cost-effective while still preserving eye-pleasing results.

The aim of this thesis is to create a neural network-based solution that can be used to control pan, tilt and zoom of a camera. The solution must be able to keep an individual performing gymnast in frame. Gymnasts perform on a standard floor area that is used for rhythmic gymnastics. The solution must be fast enough to follow a gymnast in real time.

One way of formulating this task is to consider filming a performing gymnast to be a human detection problem. Human detection is a well researched area. Modern approaches for human detection are largely based on deep convolutional neural networks. However, pre-trained models often fail to detect people doing gymnastics poses (Figure 1). Moreover, human detection solutions are trained to detect multiple people, yet in scope of this thesis we only need to detect and track one performing gymnast. This adds a new problem: how to distinguish a currently performing gymnast from people on the background. Another problem is that human detection is quite slow. For example, Faster RCN Inception V2 COCO Model, which gives a fair trade-off between accuracy and speed for GPU accelerated environments, works approximately at only 4 frames per second on GPU and 7-8 times slower on CPU [1].

There are some good object tracking solutions like GOTURN (Generic Object Tracking Using Regression Networks), which is a neural network based solution that takes some object of interest marked on one frame and locates this object in subsequent video frames [2]. GOTURN was trained using a collection of videos and images with bounding box labels but no class information. The problem is that the solution is too generic and requires an object of interest to be selected first. In my case, the object of interest is always the currently competing gymnast, the solution should be fully automated and is supposed to detect the object of interest automatically as soon as the gymnast enters the floor area.

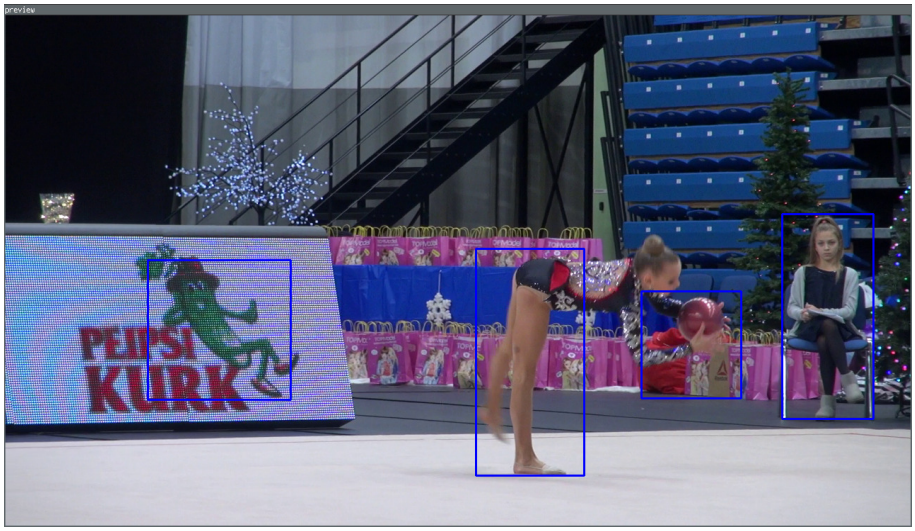


Figure 1. Pre-trained models often fail to properly detect a gymnast (V2 COCO Model).

Even if I used rhythmic gymnastics data for training or tuning, and aforementioned solutions were able to generate a perfect bounding box around gymnasts, it would still leave the camera motion planning problem unresolved. Motion planning involves figuring out pan and tilt motor speeds and zoom level change that need to be passed to the hardware. In other words, this approach just shifts the problem into a different domain.

Another interesting approach is to use a master-slave setup which consists of a static camera and another Pan-Tilt-Zoom (PTZ) camera. To create a correspondence between master and slave, a mapping between cameras usually has to be done. This can be achieved through camera calibration, which needs to be redone if cameras were moved. The method proposed by Reis et al. aims to provide the correspondence between master and slave cameras without performing a traditional calibration [3]. The method records corresponding points between master and slave cameras that will be used as input in a learning phase to perform target tracking in real time. After the corresponding points are found, the method estimates pan, tilt and zoom parameters so that the target is in the center of the PTZ camera view. The downside of this method is that it requires at least two cameras, which makes the setup more expensive and less portable.

Simple image processing techniques were not considered in this thesis because this approach was implemented in another thesis and the result had some downsides [4]. The biggest problem of that solution was the inability of the system to predict future camera motion and zoom level change in order to start the motion proactively. Another problem was the inability to implement both tilt and zoom simultaneously. Moreover, as the

solution is sensitive to color changes, it is sometimes required to reconfigure the system during a competition if lighting conditions have changed.

In this work, I use a one-camera setup and try to skip the human detection part, so that given the provided frame or sequence of frames, the neural network can predict a required camera motion and zoom directly (end-to-end approach). Recent research has shown that a convolutional neural network can learn to detect secondary features without the need of explicit labels for the position of the object of interest during training [5], [6]. Instead of expensive PTZ systems being sold on the market, I use a low-cost camera motion system that was developed by a Computer and Systems Engineering student of TalTech as part of their Master's thesis [4]. The output of my neural network is passed to the camera motion control system. Predicted pan and tilt values affect motor speeds and zoom value is passed to the camera through LANC protocol. This protocol allows the software to operate such functions as power on/off, zooming in/out, etc. [7].

1.1 Author's contributions

In this thesis, I created a manual annotation tool which was later replaced by another solution created by me. This new solution is able to annotate images in a fully automated way. Videos used for creating the dataset were also recorded by me. After the dataset was generated, I applied some balancing and augmentation techniques in order to prepare the data for training. When the input data for a neural network was ready, I created, trained and adjusted a CNN and an Long Short-Term Memory (LSTM) networks. The training was followed by a performance evaluation of the CNN, based on videos that I recorded in a new environment. I then compared the result in this new environment to a human operator. I also created a simulation tool where I compared the performance of the trained CNN and LSTM models.

2 Setups used

In this chapter, hardware and software setups and the communication between them are described.

2.1 Hardware

Hardware setup consists of a camera motion control system, a camera, and an image capture device. To make the neural network training process faster, GeForce GTX 1070 graphics card with CUDA was used. For running the system (predicting PTZ values in real time) a laptop with the following specifications is used: `Intel(R) Core(TM) i7-3520M CPU`, `8 GB RAM`, using `x86_64 GNU/Linux` as the OS. The solution is expected to run on any modern laptop.

2.2 Pan and tilt system

I am using an existing pan and tilt system developed in the scope of another Master's thesis [4].

To interface with the pan and tilt system, zero-configuration (Zeroconf) networking is used. Zeroconf is a set of technologies that allow services on the network to be discoverable without special configuration servers or manual configuration [8]. ZeroMQ is used as a messaging library [9]. ZeroMQ allows publish/subscribe pattern without a dedicated message broker. A publish/subscribe pattern is used to pass the output of the neural network to a separate node which then communicates with the hardware. Neural network outputs (pan, tilt and zoom values) are published along with a topic. Another node, which sends commands to the hardware, is subscribed to this topic. In this setup, computing the output of the neural networks does not have to be performed on the same device that communicates with the hardware, it is enough for all nodes to simply be on the same network.

2.3 Software

For training, a GPU-accelerated CUDA Deep Neural Network library (cuDNN) was installed. This library provides highly tuned implementations for standard routines such as forward and backward convolution, pooling and activation layers [10]. Python Keras library with Tensorflow backend was used for implementing and training a neural network. OpenCV was used for data preparation, extracting frames from videos, resizing images, data augmentation and also for reading live frames from a capture device. OpenCV was also used to create a simulation tool.

2.4 Required input and output

The neural network is expected to accept a video frame or a sequence of frames as an input and return pan-tilt-zoom values that are passed to the camera motion control system (pan and tilt angular speeds and zoom speed). Unlike a vanilla image processing solution which only starts reacting when the gymnast is already moving, this system should be proactive. In other words, it must be able to predict future camera motion based on the current and/or past input data. The video capture device, software and hardware have cumulative latency of around 150ms, which makes it even more important to have proactive characteristics.

3 Data preparation

One of the most important parts of this work is data preparation which consists of creating a dataset and augmenting it.

3.1 Annotation tool

The initial idea was to annotate each frame in the dataset manually. The annotation tool was written using Python GTK (Figure 2). Frames were taken from videos every second (e.g. at the interval of 50 frames in case of 50 fps videos). Then, a random unannotated frame from the dataset was picked and the annotator (human being) was supposed to submit pan, tilt and zoom values using a mouse and a keyboard. The values were based on annotator's point of view on how fast the system should pan and tilt, and how much it was needed to zoom in or out. The annotator could see a current frame and what happens in 1 second in the future. According to these images the annotator was able to submit appropriate values for the current frame. Annotations were saved in JSON files.

The biggest downside of annotating frames manually was the amount of time required to annotate large number of frames. The average time taken to annotate one frame manually was 8 seconds and the maximum number of frames annotated in a row was 270. Not only this is relatively slow, but this process cannot be sustained for a long time due to fatigue. Unlike other annotation tasks, this process is harder because it requires human judgment and prediction, not just marking an obvious feature on an image. Another problem is that manual annotations are subjective and may also be inconsistent.

Moreover, this approach would not let me experiment with different time values in the future. Thus, if I wanted to annotate frames according to what happens in less than or more than 1 second, I would have to annotate every frame again.

After over 3000 frames from different events were annotated (during 52 sessions that took cumulatively over 6 hours of work), the resulting dataset was used for the initial CNN training. The results were promising, but the idea to annotate images manually was abandoned and a new, fully-automated approach was implemented.

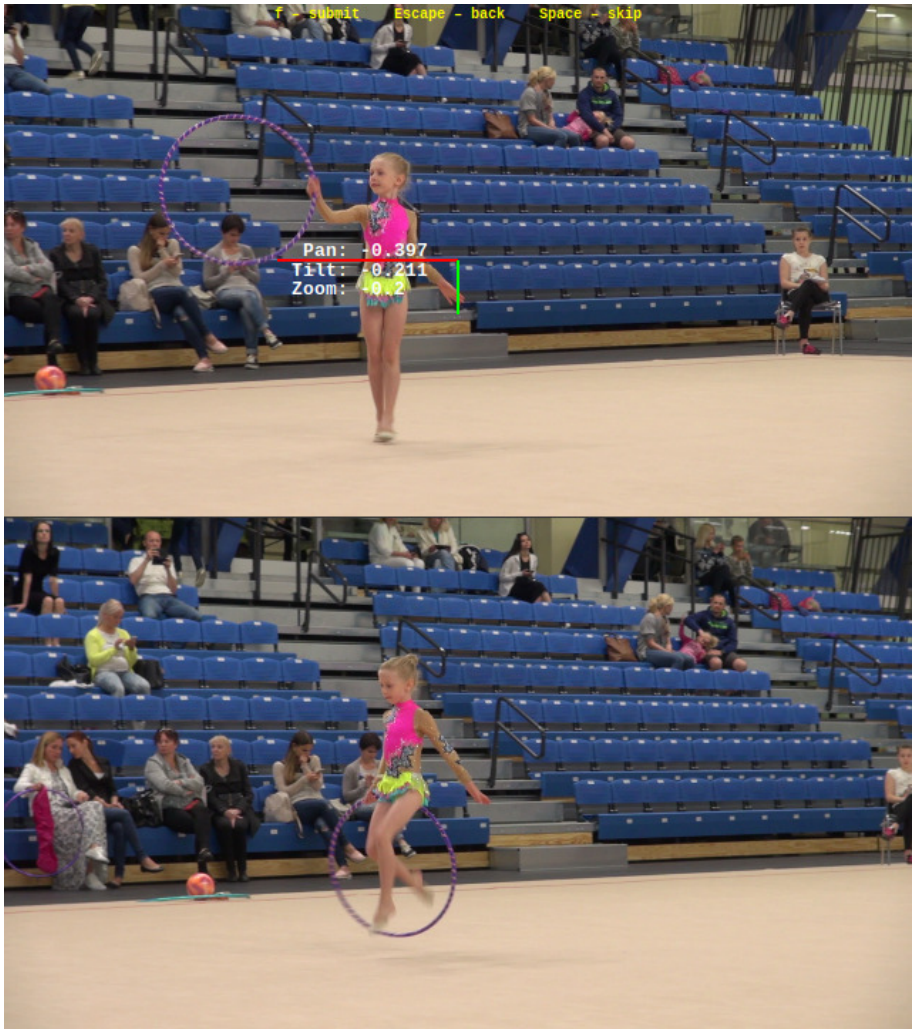


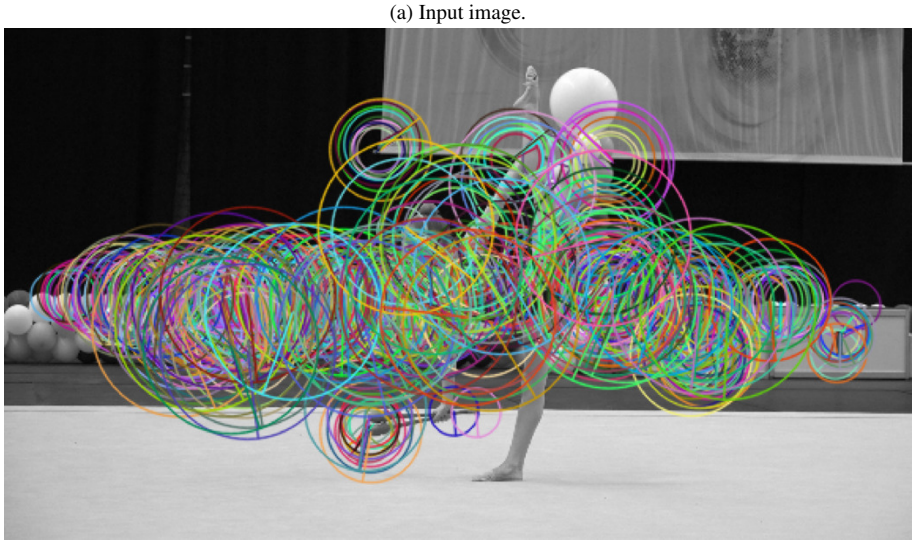
Figure 2. Manual annotation tool that was created for this thesis. Pan, tilt and zoom values change when the annotator moves the mouse. A keyboard is used to skip a frame, go back to a previous frame or save values.

3.2 Automated annotation process

The new approach is based on image registration [11]. During this process, a single common coordinate system is found for different images.

In order to create an annotation automatically, features are found in two images using Oriented FAST and Rotated BRIEF (ORB), an algorithm that was introduced by Ethan Rublee et al. [12]. Speeded-Up Robust Features (SURF) [13] and Scale-Invariant Feature Transform (SIFT) [14] algorithms, which are relatively popular, were not considered because of patent issues [12], [15], [16], [17]. Moreover, SIFT and SURF implementations were removed from the default installation of OpenCV 3 [17].

(a) Input image.



(b) ORB feature detect.

Figure 3. ORB algorithm visualization.

After finding features with ORB (Figure 3), features are then matched between the images, as shown in Figure 4. After matching is done, camera parameters are estimated roughly and later refined by trying to minimize the error between matched features.

In OpenCV, image registration is available as part of the image stitching pipeline [18]. To perform image registration in order to get pan, tilt and zoom values, a C++ program was used. The program is based on the detailed image stitcher example from OpenCV which was modified to only run steps that are required for image registration [19]. After image registration is done, I am able to get pan and tilt values by examining camera extrinsic parameters, and get zoom change by examining camera intrinsic parameters. This allows

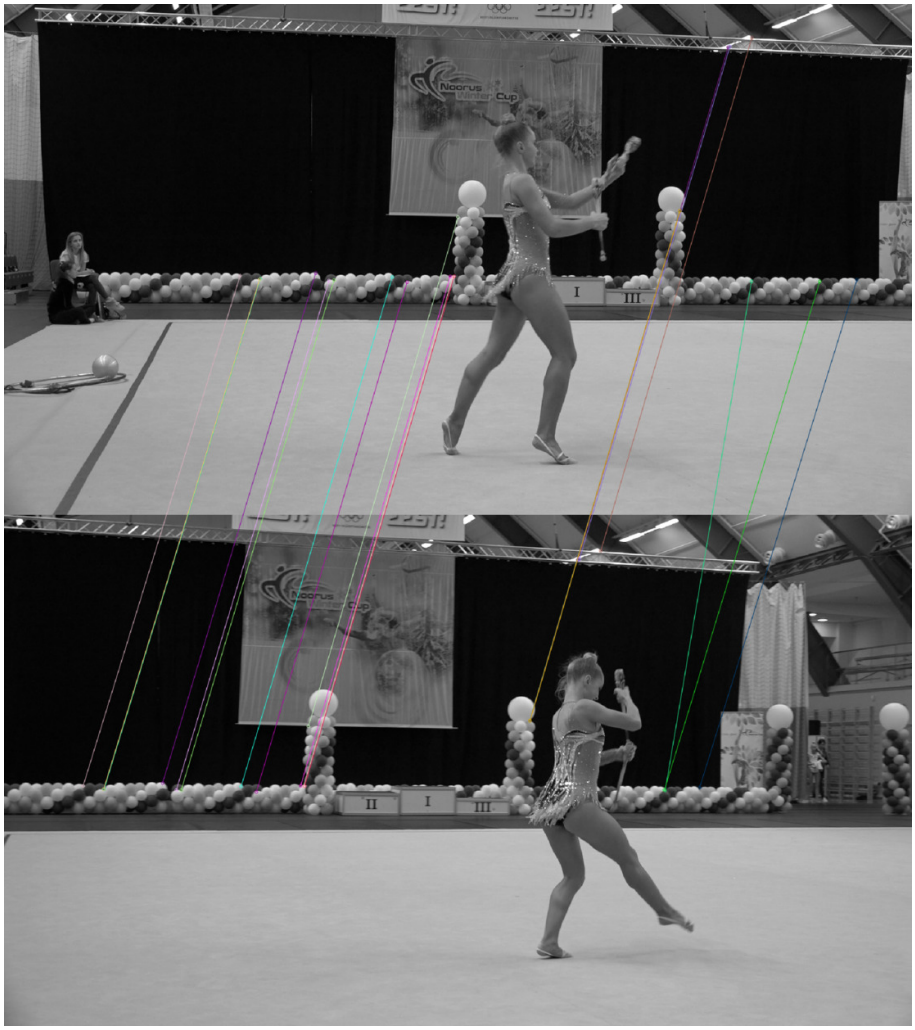


Figure 4. Matching features detected by ORB. This figure shows 20 best matches.

to annotate a large amount of data fully automatically and the annotations do not depend on the annotator's point of view. On the other hand, the algorithm does not produce absolutely accurate values. Yet, produced values are consistent and usable for training a neural network. The average speed of generating annotations for one frame using this approach is approximately 3 seconds. Not only this is 2.5 times faster than annotating frames manually, but also the process can be run without any interruptions caused by annotator's fatigue.

3.3 Dataset

The dataset consists of videos recorded by me during several events in different Estonian sport facilities. The total length of the used videos is over 85 hours. To generate a dataset, a Python script was written. The script accepts a video stream and creates datapoints every second of a video if image registration succeeds. Not only the annotated frame, but also some previous frames are saved in order to create a neural network that can use context. As context frames may be duplicated across different datapoints, I used jdupes to minimize the size of the dataset on the storage device [20].

Before feeding frames into a neural network, they need to be resized. Bigger images need more computational operations per layer and have higher memory requirements. For my neural networks, I scaled down the width and the height of every frame 10 times and kept the original ratio of 16:9. The size of a resized frame is 192x108 pixels, and its area is ¹⁄₁₀₀ of a Full HD frame (1920x1080 pixels). The total size of resized images and corresponding JSON annotations is 111.3 GB after hardlinking with jdupes.

3.4 Data filtering

The total number of datapoints in the dataset is 186130. Before using the dataset, some datapoints need to be filtered out. A sanity threshold for each parameter (pan, tilt and zoom) was specified. These thresholds help to filter out non-realistic values of camera rotation and zoom. Upon inspection, it is clear that such datapoints are usually created when someone walks in front of the camera, or if the camera was being moved to another location during recording (Figure 5).

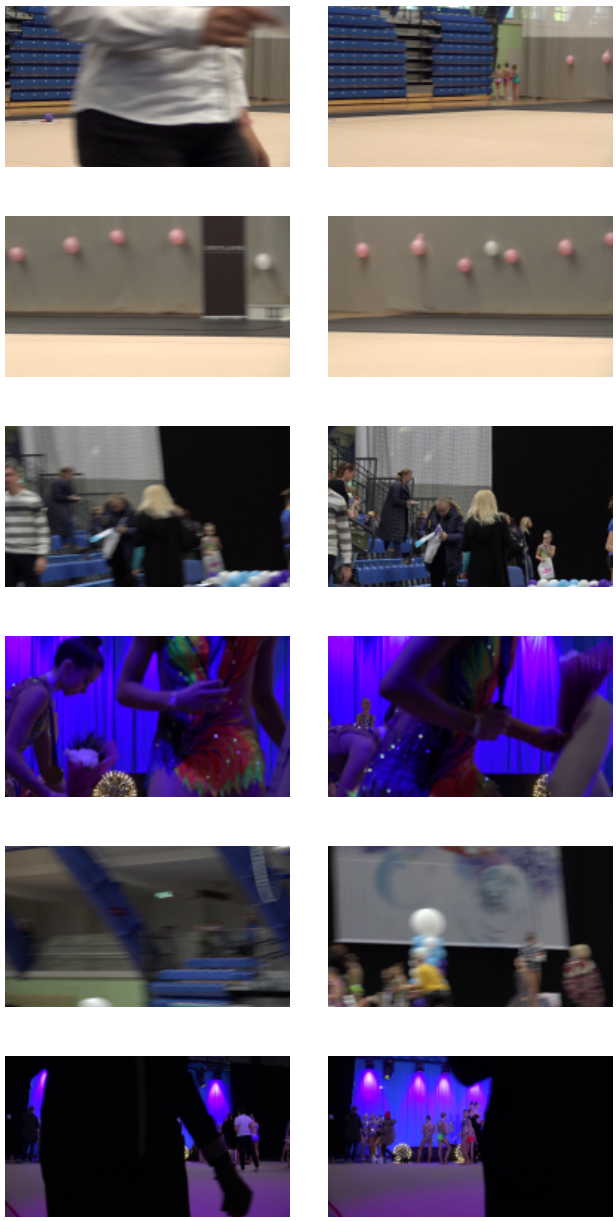


Figure 5. Sample frames that were removed from the dataset during sanity filtering. Left: annotated frames. Right: frames following in 0.5 seconds.

Next, zero threshold filtering was applied. As a result, datapoints with no camera movement were removed. Most of the detected frames were recorded during warm-up and awarding, when camera was not actively operated (Figure 6).

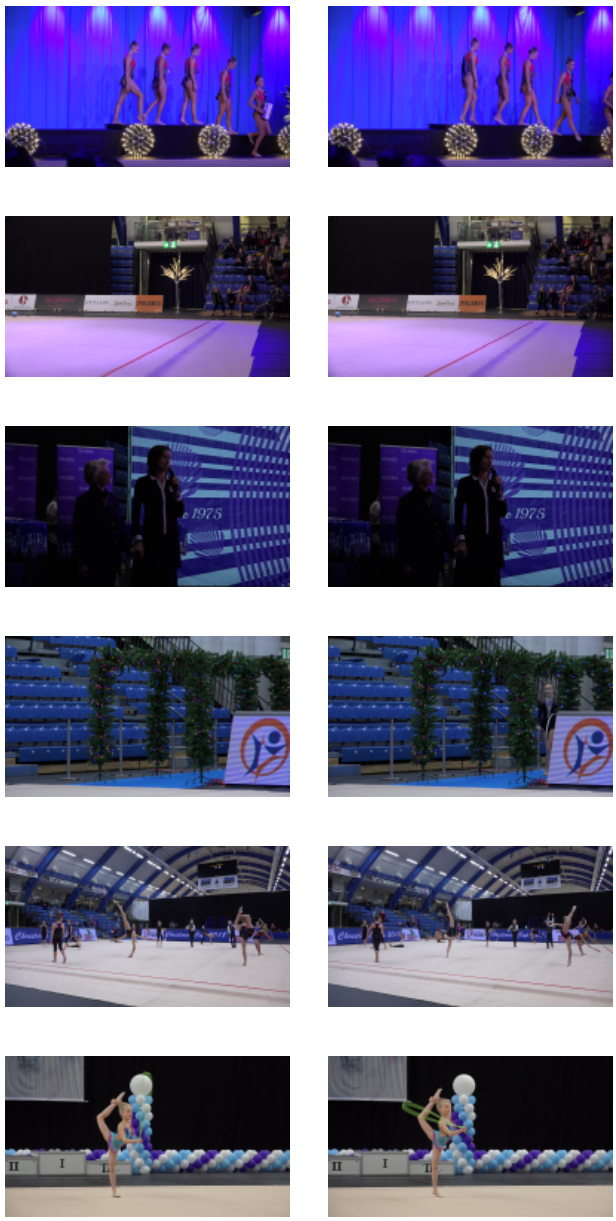


Figure 6. Sample frames that were removed from the dataset during zero filtering. Left: annotated frames. Right: frames following in 0.5 seconds. While zero filtering may also filter out some frames recorded during performances, it efficiently removes frames that are useless for training.

After removing non-realistic and zero values, the number of datapoints was reduced from 186130 to 139628 (Figure 7). Next, the dataset was split into buckets of a certain width. For example, pan values were split into buckets of width $0.5\,^{\circ}/\mathrm{s}$. To balance the dataset, bucket capacity was specified. After pan values were balanced, each bucket consisted of up to 8000 samples (Figure 8). The same approach was applied to tilt and zoom.

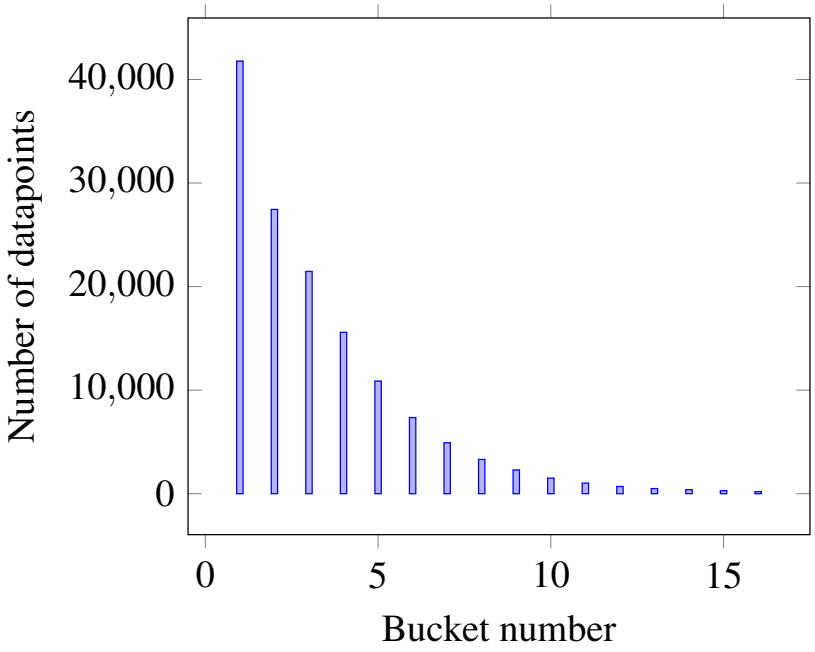


Figure 7. Pan dataset before balancing.

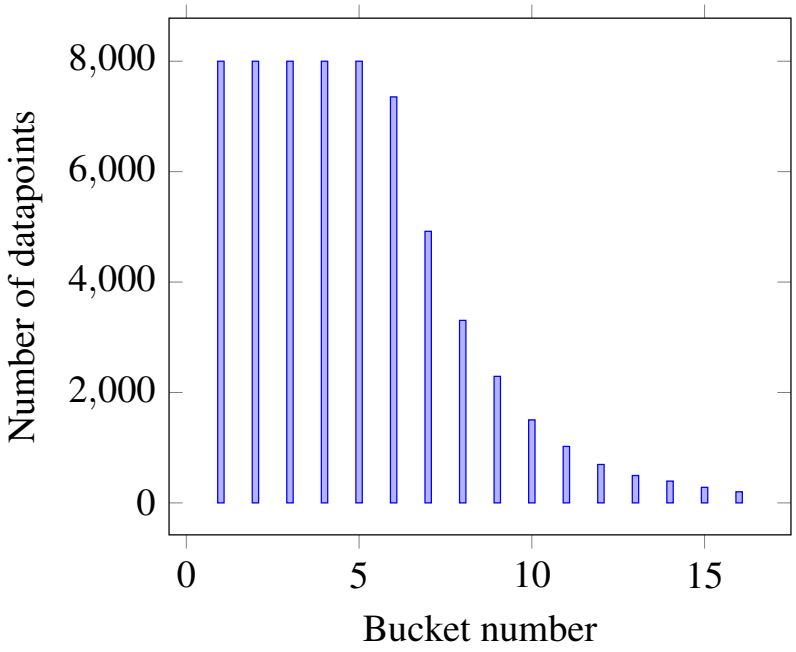


Figure 8. Pan dataset after balancing.

3.5 Data augmentation

To avoid data overfitting, some augmentation techniques were applied. Data augmentation is a process of taking existing training data and deforming it to produce new training data [21], [22], [23].

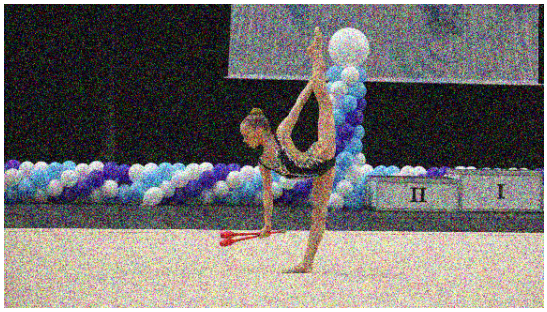
Augmentation can be done either offline or online. In the first case, data augmentation is performed beforehand and a new extended training set is saved. Another option is to enlarge a dataset during the training process by generating the augmented datapoints on demand. In this work, online augmentation was used. Next filters are randomly applied to datapoints during the training process (Figure 9):

- Horizontal flip
- Gaussian noise
- Color shift
- Brightness shift
- Blur

The effect of noise, color shift, brightness shift and blur filters is random each time they are applied. This helps to make datapoints as different as possible and reduce risk of overfitting. The order of effects is important and another order would produce different output, see Figure 10.

(a) Original image.

(b) Flip.



(c) Noise.

(d) Color shift.

(e) Brightness shift.

(f) Blur.

Figure 9. Effects used for data augmentation. Filters are applied aggressively for demonstration purposes.

(a) All effects applied in the described order.

(b) Same effects applied in reverse order.

Figure 10. Different order of filter application produces different output.

4 Convolutional Neural Network

The next step after data preparation is a neural network implementation and training. To predict a required camera motion and zoom, I started with implementing a convolutional neural network.

4.1 Background

CNN is a type of neural network. A neural network is a collection of connected units which can pass a signal from one unit to another. However, if we use regular neural networks for images, they will be very large due to a huge number of connections, resulting in a very slow training process and low performance when using the model. A CNN has one or more layers of convolution and maxpooling which reduce the complexity of the network. Typically, a CNN is followed by fully connected layers.

An image can be represented as an array of pixels. In my case, the dimensions of each input image are 192x108 pixels and, as they are not grayscale, there are three channels for each pixel (red, green, blue). This means that each image in the training and validation set is a 192x108x3 array. Before images are fed to a convolutional neural network, some normalization has to be done. Normalizing input values for the training samples helps to speed up the training process [24]. To normalize images, min-max normalization was used (1). As a result, each value in the array was converted from the range $[0 \dots 255]$ to $[0 \dots 1]$.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

A convolutional neural network typically consists of following layers:

- 2D Convolution
- 2D MaxPooling
- Dropout

Aforementioned layers are followed by Flattening and Dense layers.

Convolutional 2D layer uses a convolution kernel that is convolved with the layer input to produce the output (Figure 11). Kernel size and stride can be different. The size can be any dimension of a rectangle. Stride is the number of pixels moved per every output value. The learned filters of a CNN detect features or patterns in images. The deeper the layer, the more abstract the pattern is [25].

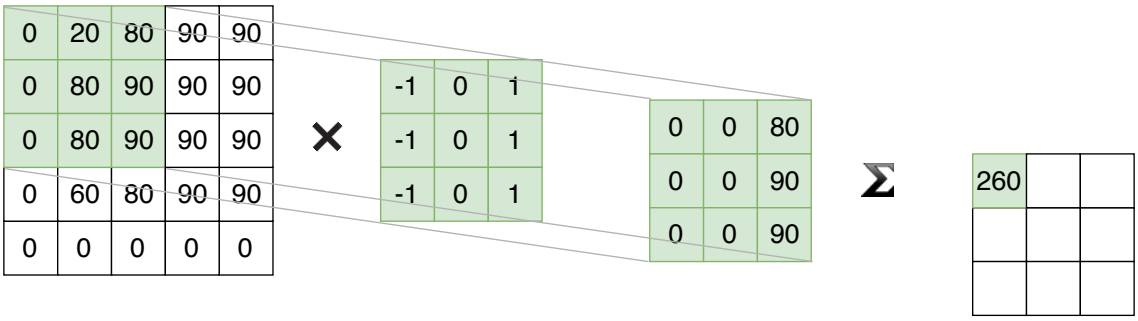


Figure 11. Illustration of applying a kernel.

Convolution layers are usually followed by pooling layers. Pooling layers reduce the size of the image across layers by sampling [25]. During maxpooling, a window moves across a 2D input space, and the maximum value within that window is the output (Figure 12).

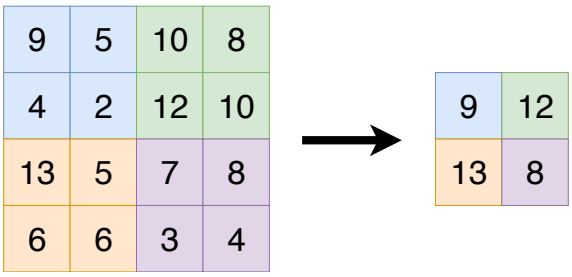


Figure 12. Illustration of maxpool with 2x2 window and stride 2.

Dropout layers are used to prevent the neural network from overfitting. Overfitting occurs when a network models the training data too well and fails to generalize [26]. The term "dropout" means to temporarily remove a unit from a network, along with its incoming and outgoing connections, see Figure 13 [27].

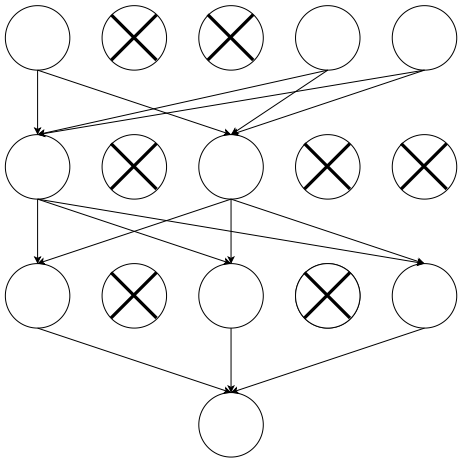


Figure 13. Illustration of applying dropout on a neural network.

A flattening layer takes a multi-dimensional volume and flattens it into a one-dimensional array prior to feeding the inputs into dense (i.e. fully connected) layers [26]. The idea of flattening is shown in Figure 14.

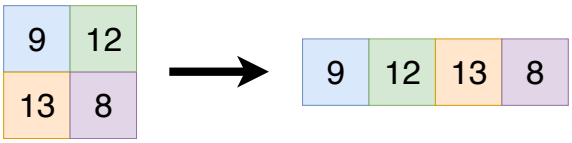


Figure 14. Illustration of flattening.

4.2 Architecture of the created network

The exact structure of the created neural network is shown in Figure 16.

The created network consists of a convolutional layer, followed by maxpooling and dropout layers, repeated three times with a different number of filters applied on convolutional layers. These are followed by three fully connected layers, with a dropout layer after each of them.

I used convolutions with a stride of 2 pixels in the first convolutional layer and convolutions with a stride of 1 pixel in the second and third convolutional layers. The size of

the kernel applied to find features was 4x4 pixels. The number of filters applied was 16 on the first convolutional layer, 32 on the second convolutional layer and 64 on the third layer. After each convolutional layer, an activation function is applied. ReLU (Rectified Linear Unit) was used as an activation function. This is the most used activation function for CNNs. ReLu activation function transforms the input to the maximum of either zero or the input itself (Figure 15). In other words, f(z) returns zero when z is less than zero and f(z) is equal to z when z is above or equal to zero. With this activation function, the more positive the neuron, the more activated it is.

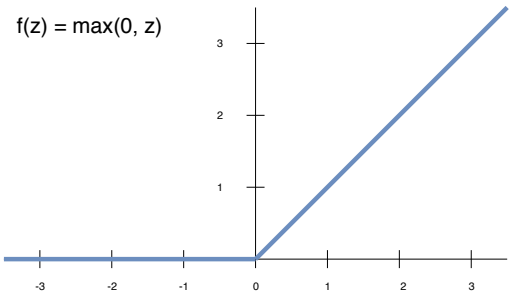


Figure 15. ReLu activation function.

Convolutional and maxpooling layers are followed by three fully connected layers. The first two fully connected layers consist of 50 neurons each, the third one consists of 20 neurons and the last one is just one neuron which returns the output.

Mean Absolute Error (MAE) was used as a loss function (2). Absolute error is the absolute value of the difference between the actual value and the output of the neural network. The smaller the mean absolute error is, the better the neural network model.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|e_t| \tag{2}$$

Widely used Mean Squared Error (MSE), which is the sum of squared differences between the actual value and the output of the network, was not used because in case of this network we do not want to penalize for bigger errors as we do not expect annotated values to be extremely accurate.

To achieve the minimum possible error for a model, hyperparameters have to be tuned. To improve the accuracy of my models, I tuned such hyperparameters as number of epochs, dropout value and batch size. These are the parameters that had the biggest influence

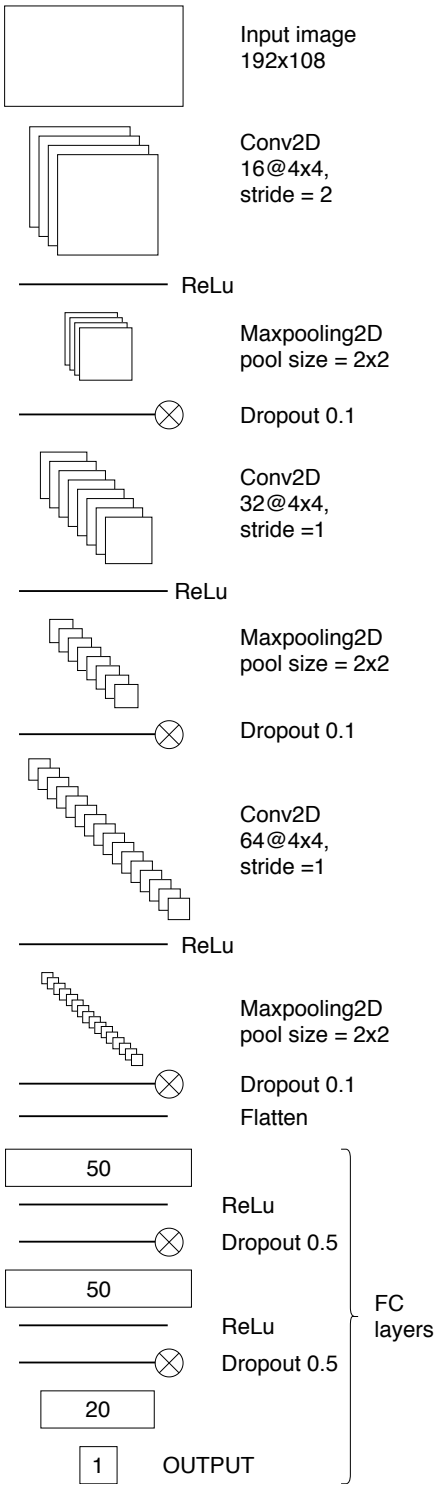


Figure 16. CNN structure. In the illustration, the number of filters is scaled down four times, but the ratio is kept to demonstrate how the number of filters increased on each convolutional layer..

on the neural network validation loss. Tuning other hyperparameters, such as number of layers and number of filters did not result in a significantly different loss.

The number of epochs determines how many times the training has to go through the full training dataset. If the validation loss is still decreasing at the end of all epochs, the number of epochs can be increased [25]. The optimal number of epochs was 98 for pan model and 175 for zoom model. The tilt model was not properly trained due to the lack of variation in tilt values in the existing dataset, therefore only pan and zoom models will be analyzed further in this work. However, as tilt is not critical for this type of a system, the setup can be successfully operated only by panning and zooming.

The process of pan model training is shown in Figure 17. Larger amounts of epochs were also tried for the pan model, however there was no measurable improvement after 100 epochs.

Dropout is most commonly used on each of the fully connected layers before the output, as it was proposed by Hinton [28]. However, recent research has shown the efficiency of using dropout in a CNN after a downsampling operation [29]. In my tests, without applying dropout at all, the mean absolute error of the model was 1.68 after 100th epoch. With dropout applied after fully connected layers it was only 1.55. However, the smallest error, 1.47, was achieved with dropouts applied after both maxpooling and fully connected layers.

Before training, the dataset is split into batches. The batch size is the number of samples that will be passed through the network at one time. Typical batch size ranges from 32 to 512 datapoints [30]. For my CNN, I used a batch size of 64 samples. The higher the batch size, the more memory is required [25]. Even though there was no issue with memory usage, smaller batch sizes result in neural networks that generalize better. The downside of using small sizes is that the training process is slower because of the data passing overhead.

During training, a model is saved after every epoch in hdf5 file, which contains [31]:

- the architecture of the model, allowing to re-create the model
- the weights of the model
- the training configuration (loss, optimizer)

- the state of the optimizer

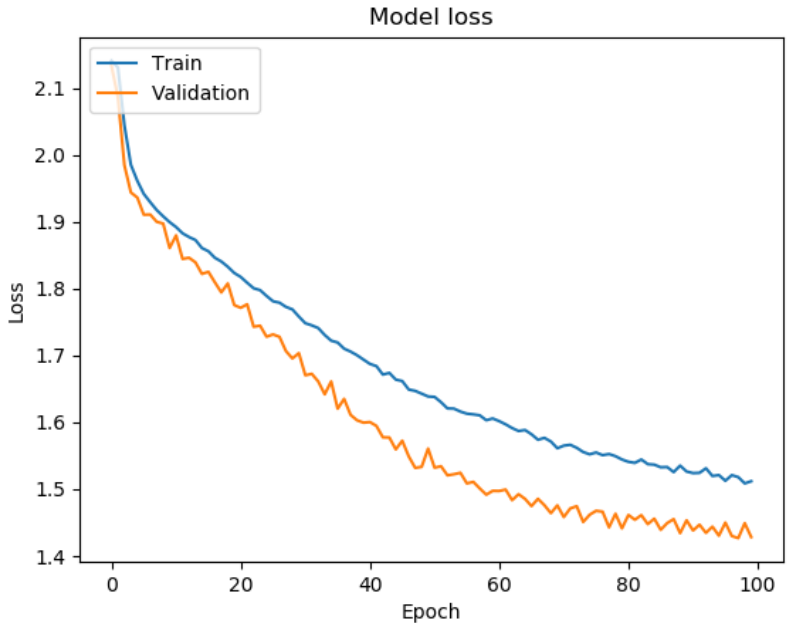


Figure 17. Pan model training.

Saving a model after each epoch allows to choose the most accurate model, stop the training and resume where it was left off.

4.3 Interpreting the output

To apply trained models to an input frame, a prediction script reads a video stream from the capture device, scales input frames down and passes those to the models. Reading and resizing frames is done using OpenCV. Outputs of neural networks are pan, tilt and zoom values. Pan and tilt values represent angular speed (degrees per second) and zoom values stand for focal length change (mm per second). In this part, data augmentation is not done and input frames are only scaled down before applying a model.

For every parameter (pan, tilt, zoom) a different model is used. While it is possible to create a single combined model, this approach is not used because multiple models offer higher flexibility. This means that any model can be replaced by another model or a different solution.

As CNN processes each frame independently, the output may be too noisy to be used to

control the hardware. This is the reason why some smoothing needs to be done before values are passed to the hardware. To achieve this, exponential smoothing was used (3). This technique allows to remove noise from time series data. To smooth the output of the neural network, the decay coefficient for each model type was specified. Decay coefficient α is a value between 0 and 1. Decay coefficients were chosen experimentally. The coefficients used were 0.2 for pan and tilt and 0.1 for zoom.

$$S_t = (1-\alpha)S_{t-1} + \alpha Y_t \tag{3}$$

Another smoothing method, a moving average, was also tried. This method is based on computing the average of last n values. A moving average was not efficient for the task because all values have the same weight. In case of the exponential smoothing, the last value can have a bigger influence.

5 Performance analysis

Before training, all datapoints were split into two sets: a training set and a validation set. I used 80% of the data for training and the rest for validation. Mean absolute error was used to evaluate how accurate a model was. The lowest mean absolute value I managed to achieve was 1.42 for pan and 236.90 for zoom. The MAE was a good measure to evaluate if one model was better than another one, but to present the efficiency of the neural network, manual evaluation was used.

5.1 Manual evaluation

To evaluate the results, 15 performances were recorded during the "Elegance Cup" competition held in March, 2019. Recorded videos were split into frames that were manually evaluated by a human operator. The length of each routine is approximately 90 seconds. Frames were extracted every 3 seconds, therefore there were about 30 frames generated for every routine, see Figure 18. Three seconds is a reasonable amount of time to recover the camera angle in case the system has lost track of a gymnast.

Another metric was how many times the performer was lost by the system. In other words, how many times it was needed to switch to another camera or to operate the camera manually until the gymnast was back in frame, or to wait until the system starts tracking again by itself.

Table 1 shows that the tracking was perfect from the beginning to the end for one third of the videos recorded to test the system. There were three main reasons why the tracking was lost:

- The podium on the background
- Gymnasts on posters on the background
- Insufficient max angular speed (too low gain used for the neural network output)

The problem with the podium is related to the noisy dataset, which not only consists of frames extracted from videos of rhythmic gymnastics performances, but also includes awarding ceremony and warm-up videos. This may be the reason why the neural network starts tracking the podium instead of the performing gymnast. This problem can be solved

Figure 18. Frames generated for manual evaluation.

by creating a better dataset which does not contain frames that are not useful for training.

The posters with gymnasts on them can also be an issue (Figure 19). However, in most cases, the performing gymnast was properly tracked even if there were posters on the background. To solve this problem, more data with gymnasts on the background should be added to the dataset.

Table 1. Manual evaluation of videos.

Video	Frames total	Bad frames	Good frames (%)	Tracking lost (times)
1	31	2	93.5	1
2	32	0	100	0
3	32	4	87.5	2
4	34	1	97.1	1
5	32	2	93.8	1
6	32	5	84.4	2
7	32	0	100	0
8	32	0	93.5	1
9	32	3	90.6	3
10	32	0	100	0
11	32	0	100	0
12	32	1	96.9	1
13	32	0	100	0
14	32	1	96.9	1
15	32	2	93.8	2

In rare cases, angular speed was not enough to follow a gymnast who was moving very fast. This could be solved by increasing the gain (scaling up the output of the neural network), but that would result in more noise when a gymnast is standing still, which can cause small camera movements from side to side. As I wanted the video to be smooth and eye-pleasing, clean output had priority over tracking at high speeds. When tracking is lost, there are at least two options. One option is to let a human operator to adjust the camera angle until the gymnast is back in frame. Another option is to switch to another automated camera, or alternatively just a static wide-angle camera. As the developed solution is used as part of a multi-camera setup, this is not considered to be a big issue.

Figure 19. Demonstration of posters on the background.

5.2 Comparison to a human operator

To compare the performance of trained CNNs to a human operator, a manually operated camera and a system operated by the CNN models were placed side by side (Figure 20).

Figure 20. A manually operated camera (left) and a PTZ system operated by CNNs (right).

5.2.1 Positioning

The difference in positioning can be seen in Figure 21. In some frames recorded by CNN models, the gymnast is slightly less centered than in frames recorded by a human. Overall, the frames look very similar.

Figure 21. Frames extracted from videos recorded simultaneously by a human operator (left) and CNNs (right).

5.2.2 Other aspects

Comparing to a human operator, zoom level change was smoother. This can be explained by smoothing being applied to the output of the neural network before passing it to the hardware. Since CNN makes predictions based on one frame only and does not see the context, predicted pan values can be quite inconsistent and may result in small jerks of the camera. Exponential smoothing helps to smooth the output of CNN but in some cases jerks are still visible while panning and this is one of the main differences that can be noticed when comparing between a human operator and the developed CNN.

6 Long Short-Term Memory network

After CNN models were successfully trained and evaluated, it was decided to try another type of a neural network. LSTM is a type of Recurrent Neural Networks (RNN) and can process sequences of data. This means, that instead of passing only one frame, we can feed multiple frames. To implement a model that combines CNN and LSTM, CNN layers are followed by an LSTM layer (Figure 22). In the scope of this thesis, LSTM was only used to train a model for panning, as pan model is the most important for the system and the performance of the trained CNN pan model is not ideal.

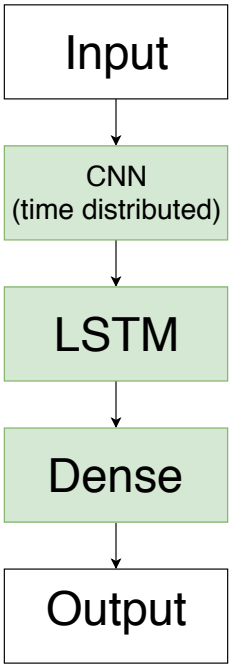


Figure 22. CNN + LSTM.

For LSTM, an extended dataset is needed. If CNN training only required one frame and corresponding annotations, for LSTM additional context frames recorded before the annotated frame are also used. To generate the context, the length of the context and time between context frames had to be specified during the generation of annotations.

Even though the process of generating the dataset is fully automated, it still requires time. To reduce the number of times I regenerate the dataset, I decided to save frames for a maximum context of 3 seconds for each annotated frame. This context length was enough to try different configurations.

To train an LSTM, following configurations have been tried:

- context of 16 frames, 3 seconds long
- context of 6 frames, 1 seconds long

First configuration resulted in a very slow training and prediction. As the system needs to work in real-time, the context length was reduced to one second, 0.2 seconds between frames. As the input shape for LSTM differs from CNN, both training and prediction scripts had to be modified by adding a new dimension for context frames. Also, in case of LSTM, data augmentation has to be applied with the same parameters to the full context, not only one frame.

6.1 Simulation tool

To test the performance of the trained CNN+LSTM models, a simulation tool was developed. For the simulation, camera calibration had to be done. To calibrate the camera, 40 pictures of a classical black-white chessboard were taken with the same camera that was used to record videos for the simulation. One of the calibration images is shown in Figure 23. These images were fed into an OpenCV calibration program, and the camera parameters were saved in a YAML file [32].

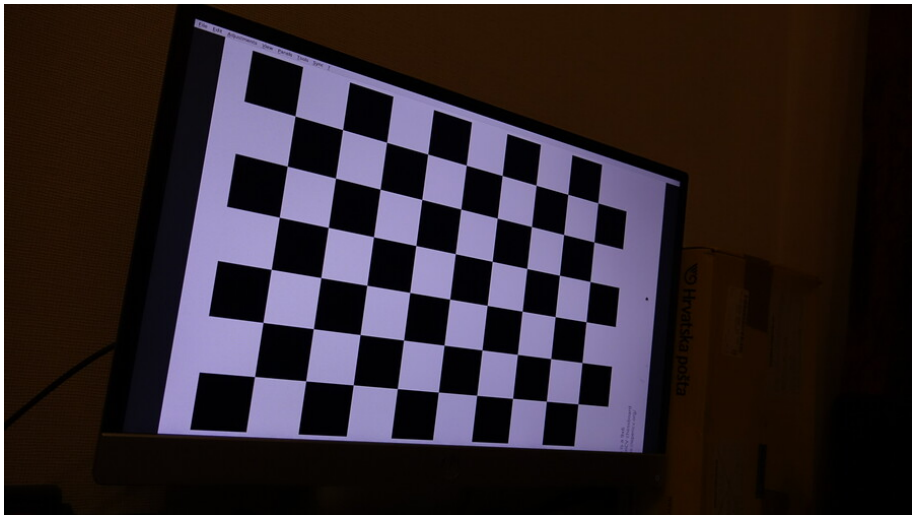
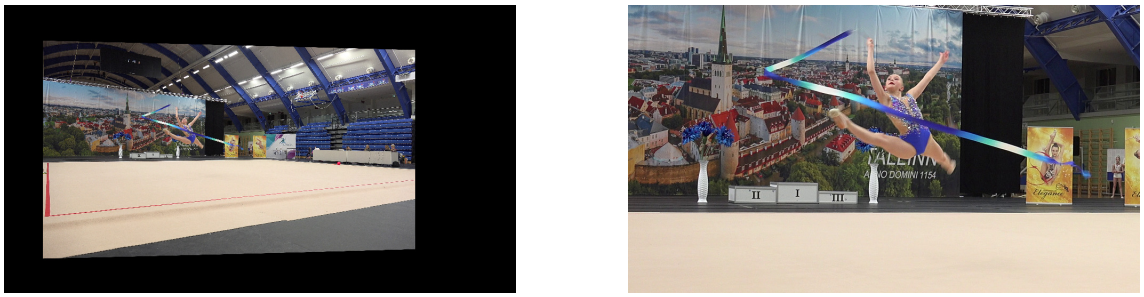


Figure 23. Camera calibration using a classical black-white chessboard.

The simulation tool was written in Python. The program reads a video stream using

OpenCV. It requires a video that covers the whole competition area and is recorded in 4K or better resolution with a static camera, see Figure 24.



(a) (b)

Figure 24. Simulation tool. a) Screenshot of a simulation tool when the camera is zoomed out for demonstration purposes. b) Demonstration of a projection of the virtual camera which is used as an input to neural networks.

During the simulation, a virtual pan, tilt and zoom are applied to the video. The output of the simulation is passed to a prediction script as a video source. The prediction script is running on another computer and receives the video stream via HDMI cable that is connected to a capture device. A setup like this preserves latency of a real system. Predictions are published on the network (publisher-subscriber pattern). The output of the neural network is then used by the simulation script to change current pan, tilt and zoom speeds accordingly, where the acceleration profiles of the real hardware are approximated. Although a cropped image is upscaled to Full HD (1920x1080 pixels) and the quality is worse than what a Full HD frame should be, this quality is enough for the prediction script which scales input images down to 192x108.

As a first step of simulating camera motion, current camera rotation angles are converted to a rotation matrix. Next step is to reproject the given video stream into the view of a virtual camera. For this task, OpenCV initUndistortRectifyMap() function is used to undistort the image and rotate the virtual camera given the rotation matrix. This function computes corresponding coordinates in the source image for each pixel in the destination image [33]. Then the projected image is generated using OpenCV remap() function, upscaled to desirable dimensions using OpenCV resize() function (in our case, 1920x1080 pixels) and displayed on the screen. The simulation tool can also be operated manually by pressing keys on the keyboard, which is used to get ground truth angles. Pan, tilt and zoom values at each frame are saved into a log file.

While the projected video from the simulation tool closely resembles a camera, it does not have some properties of a real camera. For example, the output does not have distortions or vignetting. Moreover, the motion blur in the simulation is incorrect. In case of a real camera following a gymnast, there will be less motion blur on a gymnast than on the background. However, in case of a virtual camera, the opposite is true.

6.2 Performance analysis

The first approach to training the LSTM was to train it from scratch. This means, the CNN structure was wrapped into a TimeDistributed layer and was followed by an LSTM layer and fully connected layers, and a full network was trained. The performance of this LSTM model was analyzed during a competition. The network did not track gymnasts properly, and it looked like it learned to continue the current motion of the camera regardless of where the gymnast is. This makes sense because in most cases human operators do not stop the motion abruptly, so continuing the same motion is a good rough guess. The same was observed in the simulation tool (Figure 25). In other words, if LSTM started to pan left, it would just continue panning left even if the gymnast was no longer in frame. While the model seemed to track gymnasts correctly when low gains were used (when the output of the network was artificially scaled down significantly), such performance is not satisfactory.

Better results were achieved when weights from the pre-trained CNN were loaded into a CNN+LSTM network. The structure stayed the same, but CNN weights were frozen. This model had a lower validation error (1.22) compared to a pure CNN solution (1.42). The values produced by the LSTM are less noisy and applying decay smoothing on them makes the output video even smoother. Another interesting observation made during the performance analysis is that this model leaves no negative lead room in front of a gymnast and makes the frame look more natural for the eye. This behavior is caused by the predictive capabilities of the trained LSTM network (Figure 26).

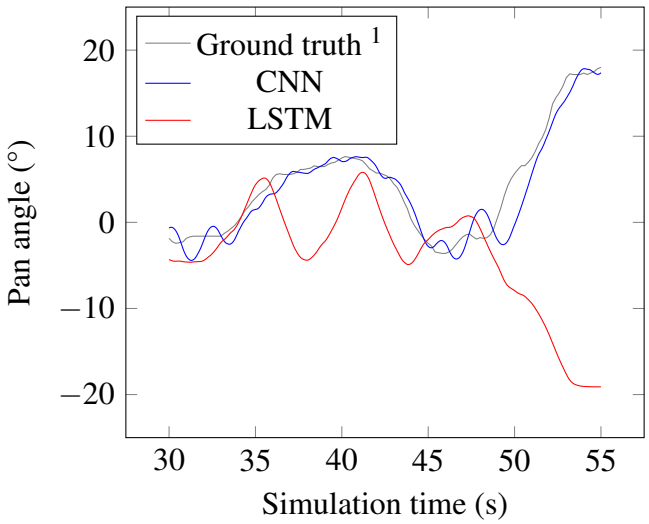


Figure 25. Results of using CNN and CNN+LSTM models (where CNN for LSTM is not pre-trained). CNN is tracking the gymnast correctly, while LSTM tends to continue the camera motion after small disturbances. This results in oscillations and eventually the tracking is lost, because the camera keeps moving even when the gymnast is no longer in the frame.

[1]Ground truth represents the manually annotated pan angle to the center of the gymnast. It is not related to how a human operates a camera or the ideal camera angle. It is shown on the graphs to demonstrate that produced camera movement is related to the actual gymnast position.

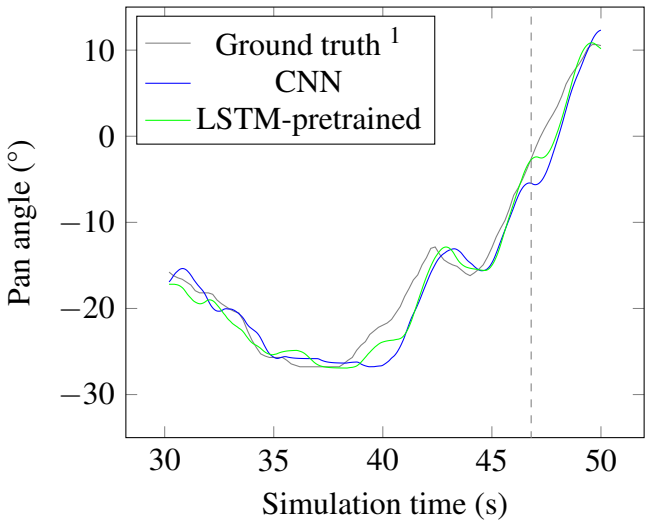


Figure 26. Results of using CNN and CNN+LSTM models (where CNN for LSTM is pre-trained). CNN+LSTM model shows its superior tracking and predictive capabilities, while pure CNN model lags behind in time. The dashed line indicates the frame that was used for Figure 27.

(a) CNN.

(b) LSTM-pretrained.

Figure 27. Same frame taken from CNN and LSTM-pretrained simulations. The gymnast is moving from left to right. LSTM manages to keep the gymnast centered even though the gymnast is moving fast and the system has latency. CNN noticeably lags behind and the video looks less eye pleasing because of negative lead room. The gray line is added for illustration purposes and indicates the center of the image.

[1]Ground truth represents the manually annotated pan angle to the center of the gymnast. It is not related to how a human operates a camera or the ideal camera angle. It is shown on the graphs to demonstrate that produced camera movement is related to the actual gymnast position.

7 Results

This chapter discusses software that was developed for this thesis and highlights the most important conclusions of the study.

7.1 Developed software

During this work, five main pieces of software were written.

The first developed tool was used to annotate pre-generated frames manually. The annotator was supposed to enter pan, tilt and zoom values using a mouse and a keyboard. The performance of the initial CNN trained on the manually annotated data was promising, but the amount of time needed to annotate frames and a possible inconsistency of values was the reason to find a better solution.

The second developed tool is used to split videos into frames and annotate those in a fully automated way. This program replaced the aforementioned manual annotation tool. It is able to generate a dataset for either a CNN or a CNN+LSTM network depending on arguments passed to the program. In both cases, a current frame and a frame that follows in a specified amount of time (normally 0.5 s) are used to produce an annotation. This process is done on many frames of a given video file using a specified time interval (normally 1 s). For CNN+LSTM, additional context frames before the current one are also saved.

The third program is used to train a neural network. This program is also responsible for data augmentation done during training. The program is parameterizable and is suitable for training both CNN and CNN+LSTM models. Depending on the specified context length, a CNN or a CNN+LSTM model will be produced. A context length of 1 frame will result in a CNN model.

The fourth program is used to predict pan, tilt and zoom values required to operate the camera using the trained neural networks. This program is also used to visualize the predicted values, see Figure 28. Smoothing of the outputs of neural network models is also done in this program. For every model, different smoothing coefficients are specified.

Finally, the last program is responsible for the simulation. It was used to test created

networks with no need to attend a competition. The simulation tool was used to visualize predictions of CNN+LSTM networks and compare them to the CNN model.

Figure 28. Visualization of the values predicted by the neural network. Red arrow: predicted pan value, blue arrow: predicted zoom value, green arrow: predicted tilt value, black arrows: smoothed values.

7.2 Conclusions

The lowest MAE was 1.42 for CNN pan model and 236.90 for CNN zoom model. These values should not be compared to each other as they are completely different. In the case of zoom, values represent focal length change speed (mm per second), while pan values represent angular speed (degrees per second).

These values alone do not give us enough information to determine whether or not the performance of the models was satisfactory. After the models were used to control the system during a real gymnastics event, it became clear that this result was good enough to successfully follow a gymnast and change the zoom level when needed. Most of the frames extracted from videos that were recorded by the system look very similar to frames from another camera that was operated by a human. There are some differences in positioning when a gymnast is moving fast. Specifically, the CNN manages to keep the gymnast in frame, but sometimes fails to keep the gymnast centered.

Tilt model was not properly trained due to insufficient tilt variation in the dataset. Due

to restrictions of the tripod that was used to record some of the videos for the dataset, a large number of the frames did not have changes in tilt values, or those changes were not big enough to be learned by a neural network. The values produced by the trained CNN tilt model are small and do not affect the camera motion much, but those values seem to be incorrect. As tilt is not critical for recording rhythmic gymnastics events and can be chosen once for a group of gymnasts of the same age and roughly the same height, the camera can be successfully operated by applying only a trained pan and zoom models. To train a better tilt model, a dataset with more active vertical motion has to be gathered.

The CNN+LSTM solution based on the pre-trained CNN showed better results for pan when compared to a pure CNN solution. Not only the MAE of 1.22 was lower than a CNN alone could achieve, but the CNN+LSTM network has predictive capabilities that help it to avoid negative lead room despite the latency of the system. In addition, unlike the CNN, the CNN+LSTM was not confused when a gymnast was turning around a vertical axis, and the direction of the face was changing rapidly.

The important step when training CNN+LSTM was to load the pre-trained CNN with frozen weights rather than train CNN+LSTM from scratch. While the MAE of CNN+LSTM trained from scratch was the lowest, the performance was not satisfactory and the network tended to just continue the current motion of the camera regardless of where the gymnast was. This may indicate that in this configuration the network has learned features that are not useful for the task.

8 Future work

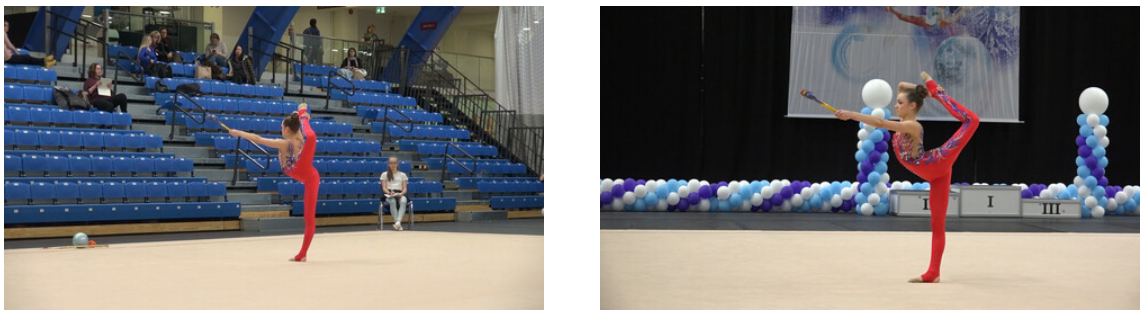
In this chapter, planned future work is presented.

8.1 Better dataset

The currently used dataset is noisy and lacks a variation in tilt values. The performance of the CNN may improve if a better dataset is created. Other approaches to annotate frames may be applied. Although the annotating approach based on image registration worked well, and CNN was successfully trained using those annotations, those are only estimated values. The most precise annotations could be created if a real data of angular speeds and zoom level change were recorded.

8.2 Multi-view setups and view switching

The next step to improve the developed system is to add support for multiple cameras, so that all cameras track a performing gymnast and the system automatically switches to the camera which has the most aesthetic view, see Figure 29. To evaluate how aesthetic a frame is, another CNN has to be created and trained.



(a) Less aesthetic view.

(b) More aesthetic view.

Figure 29. Comparison of views from different angles.

8.3 Filming between two performances

The current solution works well for tracking the competing gymnast, but there is still a problem with filming automatically between performances. The system usually stops

tracking a gymnast where they leave the floor, and when another gymnast comes, it automatically starts tracking again. This works especially well if there's a gate of some sort, because the neural network seems to know where the gate is. However, sometimes the next gymnast would come to the floor before the previous one has left it. Also, in some cases a gymnast does not return to the gate at all. These situations make the problem more complex.

8.4 Group performances

As the current solution is only trained on data gathered mostly from individual performances, another neural network is needed to automate the filming of group performances. This neural network should be able to keep all group performers in frame.

8.5 Automatic fallback to the vanilla image processing solution

If, for some reason, the neural network does not perform well, the system could automatically fall back to the vanilla image processing solution that was implemented in another thesis [4]. The challenge is to detect when this happens and to react fast enough, because that solution only works when a gymnast is still in frame.

8.6 Other camera angles

Similar network models could be trained for a close up camera that only keeps the upper part of the body or just the face in frame. Other networks can also be trained for other camera angles like a top view camera.

8.7 Other sports

The developed system can be used for similar sports, such as figure skating, acrobatics, sport aerobics, baton twirling – every sport where an individual athlete performs on a standard area. For another sport, a new dataset has to be created, but the methods of data preparation and neural network training should remain the same.

8.8 Other systems

The developed system may be used as a precursor for gymnast pose estimation and body element detection. There are neural networks that take an image of a human as an input and return estimated joint positions on the image. This can be used during trainings in order to analyze the gymnast's technique and during competitions in order to automatically evaluate difficulty and execution.

9 Summary

The aim of this thesis was to determine if end-to-end deep learning networks can be used to operate a camera motion control system during rhythmic gymnastics events. To study this problem, a dataset consisting of frames extracted from rhythmic gymnastics videos was created. Different approaches for creating annotations were tried. An annotation tool was developed to annotate frames manually, but later an approach based on image registration was implemented and taken into use. This approach annotates frames automatically, making the annotation process much faster, and does not depend on annotator's point of view. Each annotation contains pan, tilt and zoom values to be passed to the camera motion control system in order to track the currently performing gymnast. Before datapoints were fed into the neural network, some data augmentation techniques were applied.

This thesis first examined if the problem could be solved using a CNN. Results were analyzed by using trained neural networks to control the camera movement during a competition. Recorded videos were split into frames and the percent of good frames was counted. Performance analysis of the created CNN pan and zoom models showed that this approach works and can be used to capture rhythmic gymnastics events. The CNN tilt model was not properly trained due to insufficient tilt variation in the dataset. However, tilt is not critical for the system and the camera can be successfully controlled by just applying the output of pan and zoom models. To train a better tilt model, the dataset has to have more variability in tilt values.

In the second part of the thesis, a different type of neural network (CNN+LSTM) was created and evaluated. To evaluate a CNN+LSTM model and compare results between different model types, a simulation tool was created. The results of the research showed that both CNN and CNN+LSTM solve the problem, but the performance of the latter is better. Even though the video capture device, the software and the hardware have some latency, the CNN+LSTM pan model is able to track a gymnast with no negative lead room even when the gymnast is moving fast. The important step of training a CNN+LSTM was to load a pre-trained CNN with frozen weights instead of training the network from scratch. Without using existing weights from a CNN, the trained network tended to continue the motion regardless of gymnast's location.

References

[1] M. Vidanapathirana, *Real-time Human Detection in Computer Vision — Part 2*. [Online]. Available: https://medium.com/@madhawavidanapathirana/real-time-human-detection-in-computer-vision-part-2-c7eda27115c6 (visited on 03/03/2019).

[2] D. Held, S. Thrun, and S. Savarese, "Learning to Track at 100 FPS with Deep Regression Networks", 2016, European Conference Computer Vision (ECCV). DOI: 10.1007/978-3-319-46448-0_45.

[3] R. O. Reis, I. Dias, and W. R. Schwartz, "Neural Network Control for Active Cameras Using Master-Slave Setup", 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), Nov. 2018, pp. 1–6. DOI: 10.1109/AVSS.2018.8639348.

[4] A.-D. Jakimenko-Aleksejev, "Automated Camera Motion Control System for Rhythmic Gymnastics", 2019, [Online]. Available: https://github.com/RGVID-EU/RoboRG-Docs/releases/tag/1.0.0.

[5] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars", 2016.

[6] Z. Chen and X. Huang, "End-to-End Learning for Lane Keeping of Self-Driving Cars", pp. 1856–1860, 2017. DOI: 10.1109/IVS.2017.7995975.

[7] *How SONY's LANC(tm) protocol works*. [Online]. Available: http://www.boehmel.de/lanc.htm (visited on 03/06/2019).

[8] S. Cheshire and D. H. Steinberg, *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, Inc., 2005.

[9] *ZeroMQ*. [Online]. Available: http://zeromq.org/ (visited on 04/28/2019).

[10] *cuDNN*. [Online]. Available: https://developer.nvidia.com/cudnn (visited on 03/03/2019).

[11] A. Goshtasby, *Image Registration: Principles, Tools and Methods*. Springer, 2012.

[12] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF", 2011, International Conference on Computer Vision, Barcelona, Spain. DOI: 10.1109/ICCV.2011.6126544.

[13] *Introduction to SURF (Speeded-Up Robust Features)*. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html (visited on 03/19/2019).

[14] *Introduction to SIFT (Scale-Invariant Feature Transform)*. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html (visited on 03/19/2019).

[15] *Non-free functionality*. [Online]. Available: https://docs.opencv.org/2.4/modules/nonfree/doc/nonfree.html (visited on 03/05/2019).

[16] X. He, S. Luo, D. Tao, C. Xu, J. Yang, and M. A. Hasan, *MultiMedia Modeling Part II*. Springer, 2015.

[17] A. Rosebrock, *Where did SIFT and SURF go in OpenCV 3?* [Online]. Available: https://www.pyimagesearch.com/2015/07/16/where-did-sift-and-surf-go-in-opencv-3/ (visited on 04/28/2019).

[18] M. Brown and D. G. Lowe, "Automatic Panoramic Image Stitching using Invariant Features", *International Journal of Computer Vision 74(1):59-73 · August 2007*, 2007.

[19] *Detailed stitching*. [Online]. Available: https://github.com/opencv/opencv/blob/master/samples/cpp/stitching_detailed.cpp (visited on 03/06/2019).

[20] *jdupes*. [Online]. Available: https://github.com/jbruchon/jdupes (visited on 03/05/2019).

[21] J. Shijie, W. Ping, J. Peiyi, and H. Siping, "Research on data augmentation for image classification based on convolution neural networks", 2017, 20-22 Oct. 2017 Chinese Automation Congress (CAC), Jinan, China. DOI: 10.1109/cac.2017.8243510.

[22] L. Perez and J. Wang, "The Effectiveness of Data Augmentation in Image Classification using Deep Learning", 2017.

[23] E. Cagli, C. Dumas, and E. Prouff, "Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures", pp. 45–68, 2017, CHES 2017: Taipei, Taiwan.

[24] A. Levi, E. Savas, H. Yenigün, S. Balcisoy, and Y. Saygin, *Computer and Information Sciences - ISCIS 2006*. Springer, 2006.

[25] R. Shanmugamani, *Deep Learning for Computer Vision*. Packt Publishing, 2018, page 17.

[26] A. Rosebrock, *Deep Learning for Computer Vision with Python*. 2017.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research 15*, pp. 1929–1958, 2014.

[28] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", 2012, CoRR.

[29] S. Park and N. Kwak, "Analysis on the Dropout Effect in Convolutional Neural Networks", Computer Vision – Asian Conference on Computer Vision 2016. Lecture Notes in Computer Science, vol 10112, Springer, 2017, pp. 189–204. DOI: 10.1007/978-3-319-54184-6_12.

[30] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima", 2017.

[31] *Keras Documentation*. [Online]. Available: https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model (visited on 04/29/2019).

[32] *Camera calibration With OpenCV*. [Online]. Available: https://docs.opencv.org/3.1.0/d4/d94/tutorial_camera_calibration.html (visited on 04/29/2019).

[33] *Geometric Image Transformations*. [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#initundistortrectifymap (visited on 04/30/2019).