UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

Person re-identification in a top-view multi-camera environment

Supported by Ricker Lyman Robotic

Author:
Ivan PRODAIKO

Supervisor:
George BARVINOK

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2020

Declaration of Authorship

I, Ivan PRODAIKO, declare that this thesis titled, "Person re-identification in a top-view multi-camera environment" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Master of Science

Person re-identification in a top-view multi-camera environment

by Ivan PRODAIKO

Abstract

The thesis introduces the reader to the concepts of edge computing in terms of person re-identification and tracking problem. It describes the challenges, limitations, and current state-of-the-art solutions. The author proposed a pipeline for the task, launched several experiments on validating different parts of the system, and provided a theoretical explanation of the person re-identification process in the overlapping multi-camera environment.

Acknowledgements

First of all, I would like to thank Ricker-Lyman Robotics and personally George Barvinok for giving me a chance to work on such an exciting topic and supervising me throughout the way.

Secondary, I appreciate all the hard work done by Ukrainian Catholic University and Oleksii Molchanovskyi and Rostyslav Hryniv personally who made the master program possible.

Also, I am grateful to my colleagues, who gave me an opportunity to finish my master's program, especially Oleksii Khrypko, Andrey Bobryshev, Richard Buckley, Steph Moffatt, Avron Olshewsky, Vlad Ilyin Andrii Rohovyi, and Rupert Small.

Additionally, I would like to mention my classmates Vitalii Duma, Andriy Yurkiv, and Serhii Tiutiunnyk, who made my study an exciting process.

Finally, I am constantly grateful to my fiancee Mariia who consistently supported me during the study period. My life will not be so thrilling without her.

Contents

List of Figures

List of Tables

List of Abbreviations

DNN	**D**eep **N**eural **N**etwork
CNN	**C**onvolutional **N**erual **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
R-CNN	**R**egions with CNN features
MPT	**M**ulti **P**erson **T**racking
MCPT	**M**ulti **C**amera **P**erson **T**racking
YOLO	**Y**ou **O**nly **L**ook **O**nce
Re-ID	**R**e-**I**dentification
FoV	**F**ield-of-**V**iew
IoU	**Intersection**-over-**U**nion

Dedicated to my father

Chapter 1

Introduction

1.1 Motivation

Studies in the field of multi-person tracking could bring up new areas of optimization in architecture, public security, retail marketing. The creation of movement maps could show the bottlenecks of public space design.

In recent years different tools emerged and approaches for multi-person tracking that could be used to identify the number of persons on image and track them on the video stream. With the rise of deep learning (AlexNet won on ImageNat competition in 2012), the wave of CNN development brought lots of new ideas. Nowadays, the state of the art approaches introduce deep learning techniques to solve this kind of task.

The last decade was also the rise of "Big data" and cloud computing, which are the fields dealing with the massive amount of data computations and remote executions. These approaches gave rise to many tools dealing with streaming data from different devices. Some of those tools could be used for the task.

1.2 Main definitions

Multi-person tracking is the process of detection of the moving object (person) and tracking it throughout the frame sequence.

Multi-camera multi-person tracking is a process of tracking moving objects from several data sources.

Identification is a process of assigning an identifier to the detected person on a frame.

Re-identification is a process of recognizing an individual on different frames.

Cloud computing is a practice of using a remote service to perform computational operations.

Edge computing is a practice of performing computational operations on-premises (taking the raw data from smart things and send to cloud-only computed data)[1].

Detection is a bounding box selected by the object detection model.

Track is an entity representing the moving object throughout the images sequence.

1.3 Edge computing

The reasons why to use edge computing[3] rather than performing all operations in the cloud are described below:

[1] https://hivecell.io/blogs/news/six-reasons-for-edge-computing

- **Latency reduction** - the maximum speed data can move through the wires is 186,000 miles per second. Any routing point data also decrease this speed go through. These factors could create undesirable latency.
- **Bandwidth** - Sometimes, increasing the number of instances is not an option to deal with increasing load. The amount of data could be so massive, or the files could be so large that sending them using the internet could be very painful.
- **Reliability** - losing of data is unavoidable. There could factors on which humanity has no power, such as tsunami or storms. Cables and wires could be torn apart, and even massive data centers have outages where machines fail.
- **Compliance** - there are many regulations on where and how data could be stored in the cloud. The laws vary across the globe, creating additional constraints for the companies.
- **Security** - some data is considered too sensitive to move across the internet, even if encrypted and in a virtual private cloud (VPC). Mapping it to other form or converting on edge could help to solve the problem.
- **Cost reduction** - paying for transferring and storing raw data that do not generate business value is useless. The extraction of business value from the data could be done on edge, thus reducing the cost.

1.4 Challenges and limitations

- **Identity switches** - changes of object identifier throughout video stream in a predefined time duration[2].
- **Fragmentation issue** - fragmentation occurs when some detections are missed, but identity switches did not happen. This leads to tracking fragmentation.
- **Stream synchronization** - delivering messages with delay or in broken order could have a substantial negative impact on the tracking system in a multi-camera environment.

1.5 The proposed method overview

In this thesis author proposes the following approach:

- Split the tracking task into two major parts: **edge** and **cloud**. The **edge** part contains the operations executed on-premises (object detection, feature extraction), while the cloud part encompasses multi-camera tracking techniques.
- Apply deep learning techniques for object detection and feature extraction.
- Use Kafka to transfer metadata from on-premises devices to the cloud part.
- Apply modified DeepSort framework for object tracking.
- Use re-identification technique to find the same people on overlapping cameras.
- Put tracking results into Kafka topic.

[2] https://www.intechopen.com/online-first/multi-person-tracking-based-on-faster-r-cnn-and-deep-appearance-features

Chapter 2

Background / State-of-the-art

A multi-camera Re-ID task with tracking is a complex solution. It consists of 4 major parts described below.

2.1 Object detection

Object detection is a process of finding the instances of objects of a specific class in the image. Nowadays, state of the art approaches are based on deep learning.

2.1.1 R-CNN

R-CNN uses *selective search* to generate 2000 region proposals.

1. Generate initial candidates region.
2. Use a greedy algorithm to merge similar regions to larger once recursively.
3. Use formed regions to produce feature vectors.
4. Then, the feature vectors are classified using SVM.

The disadvantages with R-CNN are the time performance of the model and the huge neural network that requires lots of time to train.

2.1.2 YOLOv3

YOLO model is another neural network-based model that works way more different than R-CNN. YOLO splits the image into N×N grid and creates M bounding boxes in each cell. Then for each bounding box, the network evaluates the class probability, and if it is higher than the threshold, the model uses it to locate the object. YOLO model is exceptionally fast, which makes it acceptable for real-time object detection[5].

2.2 Object re-identification

The essence of Re-ID is to compare the similarity or distance between the features extracted from two images [15].

2.2.1 Feature extraction

The first step of Re-ID is feature extraction, which is vital for good performance. Traditional methods of feature extraction include Scale-Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF), and others. Still, nowadays state of the art approaches for Re-ID are based on deep learning (CNN, RNN)[1].

2.2.2 Distance / Similarity measurement

To compare the difference between two feature vectors could be used similarity or distance metrics.

Euclidean distance

The Euclidean distance between points p and q is the length of the line segment connecting them.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2.1}$$

Cosine similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and cosine of 90° is 0.

$$\cos(\mathbf{t},\mathbf{e}) = \frac{\mathbf{te}}{\|\mathbf{t}\|\|\mathbf{e}\|} = \frac{\sum_{i=1}^{n}\mathbf{t}_i\mathbf{e}_i}{\sqrt{\sum_{i=1}^{n}(\mathbf{t}_i)^2}\sqrt{\sum_{i=1}^{n}(\mathbf{e}_i)^2}} \tag{2.2}$$

Triplet loss

In the calculation of the triple loss, the feed data includes an anchor, a positive sample, and a negative sample, and the sample similarity calculation is realized by optimizing the distance between the anchor and the positive sample being smaller than the distance between the anchor and the negative sample [15]. After training, it could be used to compare feature vectors as related to the same class or to different.

2.3 Object tracking

Object tracking is a process of tracking moving objects on the video stream. This task is tightly connected with the Re-ID process. If the number of objects assigned with ids with the help of object tracking, these ids should persist throughout the stream.

2.3.1 Mean shift

The mean shift is one of the simplest approaches to object tracking. It uses color distribution to track objects. The algorithm consists of 4 steps:

1. Select a search window size and the initial position of the search window.
2. Estimate the mean position in the search window.

[1] https://github.com/NEU-Gou/awesome-reid-dataset

3. Center the search window at the mean position estimated in Step 2.

4. Repeat Steps 2 and 3 until the mean position moves less than a preset threshold. Until convergence achieved.

2.3.2 Kalman filter

Kalman filter is used for object tracking when the initial position of the object is known. The algorithm tries to predict the next position knowing the previous one and takes into account the change that position has some error while being estimated. Velocity is also required for predicting the next position[2].

Consider the coordinates of the moving car.

x_k - real value to be measured.

v_k - velocity on the current step.

Theoretical measurements of the next step:

$$x_{k+1} = x_k + v_k dt \tag{2.3}$$

Real measurements of the next step:

$$x_{k+1} = x_k + v_k dt + \xi_k \tag{2.4}$$

where: ξ_k - error of outside world.

Real measurements of the GPS sensor:

$$z_k = x_k + \eta_k \tag{2.5}$$

where: η_k - error of GPS sensor.

From two equations above we could find the optimal solution for predicting the next step coordinates of moving car by using the weighted sum of two measurements

$$x_{k+1}^{opt} = K_{k+1} z_{k+1} + (1 - K_{k+1})(x_k^{opt} + u_k) \tag{2.6}$$

where: K - Kalman coefficient, $u_k = v_k dt$ - controlling member.

The error of a system:

$$e_{k+1} = x_{k+1} - x_{k+1}^{opt} \tag{2.7}$$

$$e_{k+1} = K_{k+1}\eta_{k+1} + (1 - K_{k+1})(e_k + \xi_k) \tag{2.8}$$

In order to find the right value of Kalman coefficient, we need to minimize the mean of squared error:

$$E(e_{k+1}^2) \to \min \tag{2.9}$$

Base of iteration:

$$E(e_{k+1}^2) = (1 - K_{k+1})^2 (E_k^2 + \sigma_\xi^2) + K_{k+1}^2 \sigma_\eta^2 \tag{2.10}$$

where: $\sigma_\eta^2 = E\eta^2$

$$K_{k+1} = \frac{Ee_k^2 + \sigma_\xi^2}{Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2} \tag{2.11}$$

Thus, on each iteration, we are looking for an optimal Kalman coefficient.

[2] https://david.wf/kalmanfilter/

2.4 Stream processing

Stream processing is an approach to data processing, treating it as a stream of events. Typical stream processing application consists of the number of producers sending events and the number of consumer processing them. The stream processing platform has to be reliable, scalable, and resilient to changes in load. One platform that meets all these requirements is Apache Kafka.

2.4.1 Kafka

Apache Kafka is an append-only log with a configurable retention period. Kafka is very flexible and could be tuned to process hundreds of terabytes of data.[3]

Data are stored in topics that represent the streams of events. Topics are split into a number of partitions (the more, the larger the amount of data could be stored).

Producers push messages into the topic while consumers are reading from them.

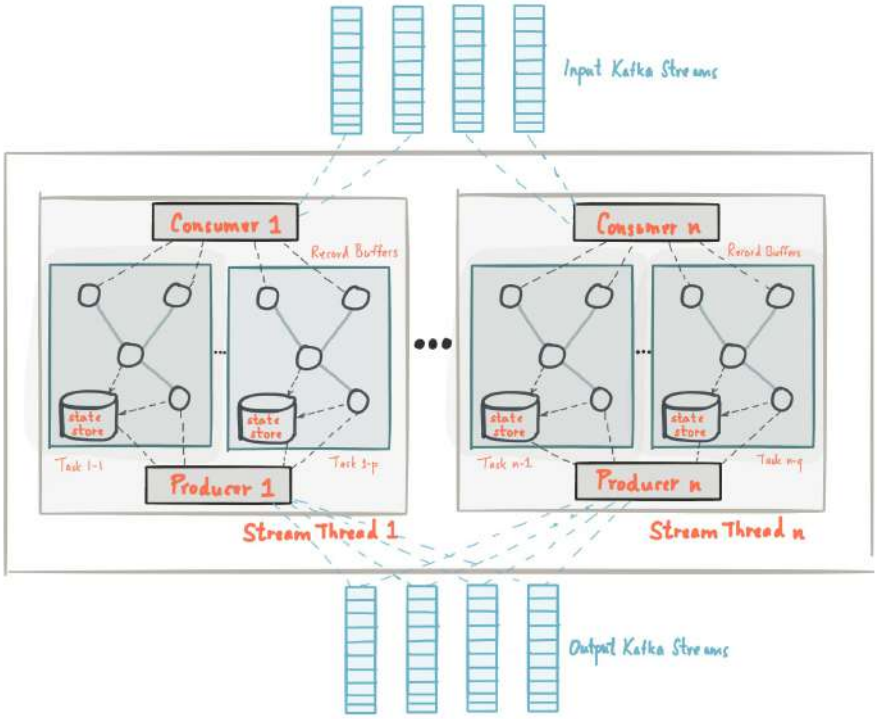

FIGURE 2.1: Kafka application overview (off. docs)

At a high-level Kafka gives the following guarantees [4]:

- Messages sent by a producer to a particular topic partition will be appended in the order they are sent. That is, if the same producer sends a record M1 as a record M2, and M1 is sent first, then M1 will have a lower offset than M2 and appear earlier in the log.
- A consumer instance sees records in the order they are stored in the log.
- For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any records committed to the log.

[3] https://kafka.apache.org/documentation/

[4] https://kafka.apache.org/intro#intro_guarantees

Chapter 3

Existing methods overview

3.1 Multi-camera Streaming Tracker using Deep Stream SDK

NVIDIA's DeepStream SDK delivers a complete streaming analytics toolkit for AI-based video and image understanding, as well as multi-sensor processing[1].

Solutions with Deep Stream SDK encompass two parts approach[2]. Object detection and object tracking are performed on-premises, while cloud component consists of multi-camera tracker and data analytics tools.

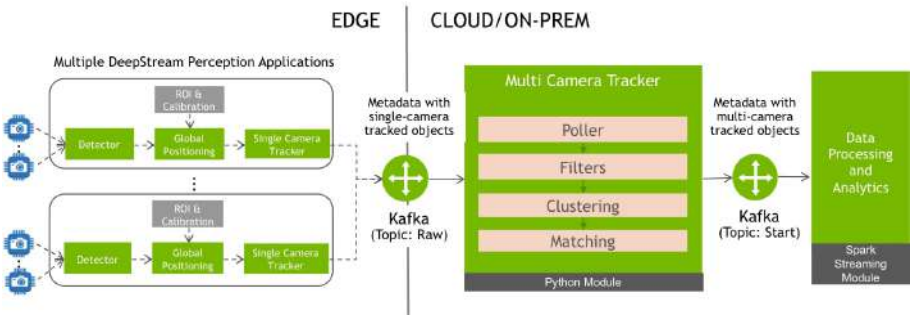

FIGURE 3.1: Architecture of multi-camera solution using DeepStream SDK

3.1.1 Multi-camera tracker overview

The most exciting part of the project is multi-camera tracker. This component of the system responds to the question of how to match two single-camera tracks and create a road map of the object moving through all cameras?

The pipeline of multi-camera tracker is described below [3]:

1. **Poller** - a step that polls Kafka to get new tracks. The polling interval is equal to the detection interval. If detection happens on 4 fps, than polling interval should be equal to 0.25 seconds.

2. **Filters** - step that remove tracks from ignored regions. Fisheye cameras which are used in solution record the 360°view. Some of the regions on these views could be ignored as noisy or unnecessary.

3. **Clustering** - a step that performs clustering of the tracks with respect to several rules:

[1] https://developer.nvidia.com/deepstream-sdk

[2] https://devblogs.nvidia.com/multi-camera-large-scale-iva-deepstream-sdk/

[3] https://github.com/NVIDIA-AI-IOT/deepstream_360_d_smart_parking_application/tree/master/tracker

- If two detections from different cameras are close to each other, they may belong to the same object[4].
- If the same camera assigned different ids to two detections than they are indeed different.
- detection with a single-camera track id belongs to only one multi-camera track id and does not change over the time.

4. **Matching** - final step matching newly arrived detections with old ones using Hungarian algorithm.

Then, results send to the resultant Kafka topic, out of which the data is retrieved by analytics tools and displayed in UI.

3.1.2 Camera calibration technique

The approach used in Deep Stream SDK only suitable for camera systems where cameras are observing a fixed FoV, meaning the camera always watching the same geo-region. Also, there must be a global map to transform a relative in-camera position to global coordinates.

To properly configure FoV with mapping to global coordinates[5] there are several steps described bellow:

1. Use an image annotation tool (authors of Deep Stream SDK suggest to use Ratsnake) to draw a polygon on one of cameras' images.
2. Use GIS tool to draw global polygon on the global map.
3. Create a mapping file containing $(Id_cam, A_{cam}, B_{cam}, C_{cam}, D_{cam}, A_{glob}, B_{glob}, C_{glob}, D_{glob})$ for each camera.
4. SDK will compute a transformation matrix.

3.2 Multi-Person Tracking Based on Faster R-CNN and Deep Appearance Features

The methodology proposed in this system are divided in several subtasks[6]:

1. Faster R-CNN for human detection.
2. CNN for appearance features extraction.
3. Kalman filter for human tracking.
4. Hungarian algorithm for tracking nearby rectangles.
5. Additional features like area, color, nearest color, and HSV.
6. Usage of face recognition when possible.

[4] https://github.com/NVIDIA-AI-IOT/deepstream_360_d_smart_parking_application/tree/master/tracker

[5] https://devblogs.nvidia.com/calibration-translate-video-data/

[6] https://www.intechopen.com/online-first/multi-person-tracking-based-on-faster-r-cnn-and-deep-appearance-features

The most interesting are the two last steps of the approach using additional features for improving the accuracy of detection.

Area of human - if one person is tall and the other is short, this will not change during the session so that this info could be used as a feature.

Color and nearest color - people do not usually change clothes while being in public places; thus, the color of clothes could be used as a feature.

Color histograms could be used as features and improve the performance of the system using cumulative brightness transfer function (CBTF) as a comparison function.

Face recognition - when person is closed enough (15-20 feet) the face recognition system [9] creates a feature map for the face. Then during the next matching phase, the feature map is used to improve detection accuracy.

Unfortunately, in the top-view tracking system, face-recognition can't be used.

3.3 Deep person Re-ID

Deep person Re-ID is a framework for person re-identification using omni-scale feature learning as a core. The framework supports multi-camera re-identification out of the box and has many datasets wrappers[7] for fast integration with them.

The key feature of the framework is its modularity, which makes it easy to extend, support, and change.

The core idea of omni-scale feature learning is to have additional attention to small features as well as the overall appearance. To match people and distinguish them from impostors, features corresponding to small local regions (e.g., shoes, glasses) and global whole-body regions are equally important[18].

[7] https://github.com/KaiyangZhou/deep-person-reid#image-reid-datasets

Chapter 4

Datasets description

Name	Extension	Task	Items	Memory
Dataset_1	mp4	-	5 videos	13.7 gb
Dataset_2	jpg + txt	object detection	3769 images	1.3 gb

TABLE 4.1: Brief datasets info

4.1 Dataset_1 / Video

The primary dataset consists of 5 hours of video from 5 cameras (1 hour per camera) with frequency 24 frames/second and size 1280 $\times$ 960. Videos were recorded in UCU dining room by Ricker Lyman Robotic.

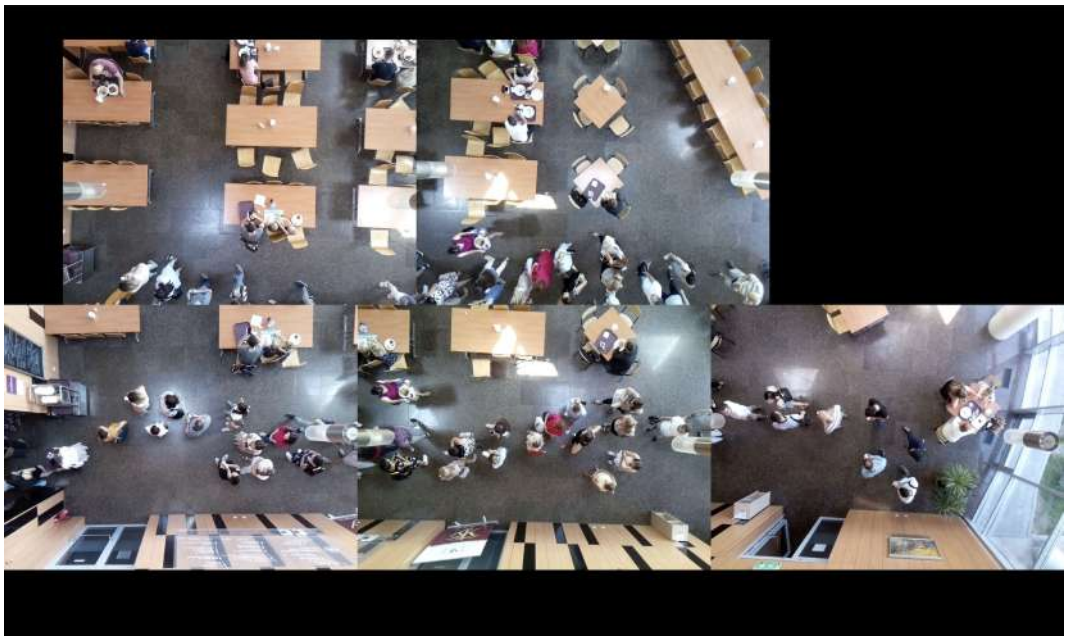

FIGURE 4.1: Merged frames from 5 cameras

4.2 Dataset_2 / Object detection

Dataset contains 4000 labeled images from 5 cameras in YOLO format from the primary dataset. For every labeled frame exists *.txt* file containing detection class (for particular case it is always *person*) and relative position of bounding box on frame.

FIGURE 4.2: Labeling process in YOLO_mark GUI tool

```
0 0.324609 0.140972 0.130469 0.126389
0 0.344922 0.491667 0.085156 0.122222
0 0.375781 0.628472 0.096875 0.137500
0 0.200391 0.436806 0.158594 0.123611
0 0.067969 0.545833 0.135937 0.116667
0 0.522266 0.518750 0.060156 0.112500
0 0.565625 0.645833 0.090625 0.127778
0 0.674609 0.388194 0.122656 0.093056
0 0.807813 0.500694 0.050000 0.151389
0 0.726953 0.504167 0.097656 0.141667
```

CODE SAMPLE 4.1: .txt file for labeled image

Chapter 5

Solution

The proposed method is a complex solution based on several existing technologies and custom implementation of the technique for overlapping multi-camera Re-ID. Method is divided into two major parts: **edge** and **cloud**.

Edge part contains all operations that could be done on-premises, operations that do not require synchronization, and could be done with the video stream of one camera - object detection and feature extraction.

Cloud part consists of all operations that require synchronization and data from several cameras - object re-identification, object tracking, tracking results streaming, analytics.

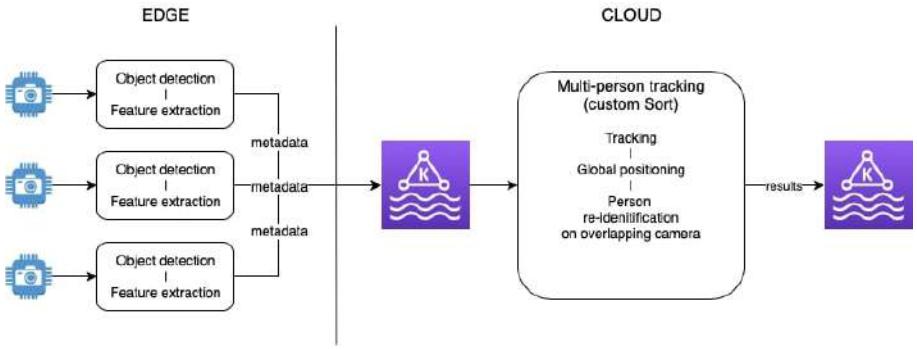

FIGURE 5.1: Architecture of proposed solution

5.1 Edge

Object detection and feature extraction happen on each camera separately and do not require synchronization. Because of this fact, the tasks in this section could be done on the premises. All GPU intensive operations concentrated here, so the hardware must have GPU support.

Technologies below are selected with performance in mind. Some of them may not be the most accurate, but outstanding performance pays off.

5.1.1 Object detection (YOLOv3)

The first step in the pipeline is object detection implemented using YOLOv3 model.

YOLOv3 could deal with the image of any size[1]. The input of the network is the image, and the output is a list of bounding boxes along with the recognized classes. Each bounding box is represented by 6 numbers (class index, top left x, top left y,

[1] https://www.cyberailab.com/home/a-closer-look-at-yolov3

bounding box height, bounding box width, confidence). As a loss function, IoU is used. This method improves predictions by making specialized bounding boxes for some aspect ratios and sizes [2].

YOLOv3 has three scales for detecting objects of different sizes. Having the strides of the network equals to 32, 16, and 8, the input image of (1280, 960) will be divided into the grid with scales (40, 30), (80, 60) and (160, 120) respectively. For each grid cell, 3 bounding boxes used. The total number of anchors is 9. After it, every bounding box gets the score. All scores below some threshold (usually 0.5) are ignored. The next step is non-maximum suppression performed over the bounding boxes overlapping each other. All bounding boxes left to form the output of the model.

Rep	Name	Filters	Patch Size/Stride	Output size
	Conv	32	3×3	256×256
	Conv	64	3×3/2	128×128
	Conv	32	1×1	
1	Conv	32	3×3	
	Residual			128×128
	Conv	128	3×3/2	64×64
	Conv	64	1×1	
2	Conv	128	3×3	
	Residual			64×64
	Conv	256	3×3/2	32×32
	Conv	128	1×1	
8	Conv	256	3×3	
	Residual			32×32
	Conv	512	3×3/2	16×16
	Conv	256	1×1	
8	Conv	512	3×3	
	Residual			16×16
	Conv	1024	3×3/2	8×8
	Conv	512	1×1	
4	Conv	1024	3×3	
	Residual			8×8
	Avgpool		Global	
	Connected		1000	
	Softmax			

TABLE 5.1: Darknet53 model architecture

5.1.2 Feature extraction

The core part of any Re-ID system is extracting a feature vector from detection. The mainstream approach for feature extraction is deep learning. The author decided to use DeepSort framework for tracking. The framework split into two parts to satisfy the needs of the multi-camera system with edge computing.

[2] https://missinglink.ai/guides/computer-vision/yolo-deep-learning-dont-think-twice/

"Deep" part of DeepSort framework

The deep part of the framework is a CNN that takes the number of detected by YOLOv3 images as an input and outputs 128 entries long feature vector. Authors of the framework claim that "one forward pass of 32 bounding boxes takes approximately 30 ms on an Nvidia GeForce GTX 1050 mobile GPU" making this NN suitable for real-time processing. [11]

Name	Patch Size/Stride	Output size
Conv 1	3×3/1	32×128×64
Conv 2	3×3/1	32×128×64
Max Pool 3	3×3/2	32×64×32
Residual 4	3×3/1	32×64×32
Residual 5	3×3/1	32×64×32
Residual 6	3×3/2	64×32×16
Residual 7	3×3/1	64×32×16
Residual 8	3×3/2	128×16×8
Residual 9	3×3/1	128×16×8
Dense 10		128
Batch and l2 normalization		128

TABLE 5.2: Architecture of deep appearance descriptor

5.2 Cloud

The "cloud" part manages the state of the application and synchronizing the events from different cameras. Between edge and cloud computational units located streaming platform responsible for streaming, replication, and temporarily storing the data.

5.2.1 Stream processing

After performing object detection and feature extraction on one camera, the data from edge computational units transferred to *Kafka topic* with the help of *Kafka client*. Each message represents one frame of the video stream. Depending on needs, some frames could be skipped to reduce the load on the system.

To transfer data and keep it consistent Avro data serialization system [3] is used. Replication of data has to be set up manually, depending on needs. In order to ensure safety when transferring data via a public network, SSL encryption must be used.

There are two possible strategies of the topic set up:

1. **All cameras - one partition**. The benefits of this approach simplicity and flexibility. If the new camera is added to the system, nothing has to be changed in configuration. The drawbacks are: reliability of the system is insufficient for long term use[4] and the ordering of messages are random in terms of *cam_id*, so forming a batch of messages that equals to the number of cameras is not an approach.

[3] http://avro.apache.org/docs/current/

[4] https://jack-vanlightly.com/blog/2018/9/14/how-to-lose-messages-on-a-kafka-cluster-part1

2. **One camera - one partition**. The benefits of this approach are scalability and reliability. Messages are partitioned by *cam_id* are stored in order where one partition storing the data from one camera. If one partition failed, it doesn't mean that the system failed. Additionally, the data from each camera could be processed separately. The drawback is configurability. If a new camera added, all of the data has to be re-partitioned, and the ordering will be lost.

Consumer polls topic for new messages and puts the data into the object tracking algorithm, which contains the state of the application.

5.2.2 Object tracking

The author used DeepSort framework for this part because of its simplicity and outstanding performance. As far as the "Deep" part of the framework was moved to the **edge** here author describes the "Sort" part only.

"Sort" part of DeepSort framework

The tracking system is implemented by using Kalman filtering and frame-by-frame data association using the Hungarian method.

Track Handling and State Estimation

Input consists of (u, v, y, h), corresponding to bounding box center position (u,v), aspect ratio y, height h. Kalman filtering has configurable constant velocity motion.

Each track could be in one of three states:

- Tentative - new tracks that have not been confirmed. To transfer to confirmed state track has to appear N times in a row. If the track is not associated N times, it is transferred to the Deleted state. N is configurable.
- Confirmed - the track that has been validated as confirmed. It is the main state. If detection for the track is missed for M number of frames than it is marked as Deleted. M is configurable.
- Deleted - tracks that will be removed from the state at the end of the iteration.

If input detection can not be associated with any of the existing detection, a new track is created.

Assignment Problem

To associate existing tracks with newly provided detection Hungarian algorithm is used. Association is done using the weighted sum of two features (squared Mahalanobis distance and cosine similarity) between predicted Kalman states and newly arrived measurements.

$$c_{i,j} = \lambda d^1_{i,j} + (1 - \lambda) d^2_{i,j} \tag{5.1}$$

where:

λ - hyperparameter regulating the influence of each of feature.

In combination, both metrics complement each other by serving different aspects of the assignment problem.

One person on several cameras case

In the dataset provided by Ricker-Lyman Robotics, each camera located in overlaps neighboring cameras, so the situation when the same person is detected on several cameras simultaneously could emerge. To deal with this kind of task author of this thesis proposes the add-on to DeepSort framework.

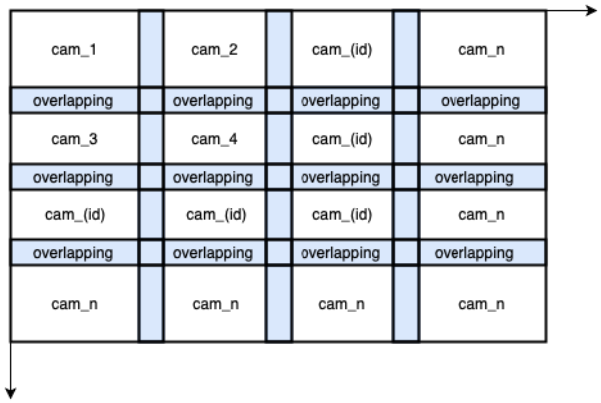

FIGURE 5.2: Top view overlapping cameras

When tracking people, DeepSort incrementally assigns ids to newly created tracks (detection that has not been associated with existing tracks). It does not have any mechanism to find tracks that could relate to the same person on by several cameras. With the proposed add-on author tries to solve this problem.

Idea explanation

If one person appears on several cameras simultaneously, the related tracks should have similar feature vectors, and these tracks have to be close to each other in absolute position. Additionally, there could not be two tracks related to the person in one camera view, so any tracks with the same *cam_id* could be ignored in comparison.

Implementation

First of all, DeepSort input has to be enriched with *cam_id* parameter that describes the relation of the track to a particular camera.

In order to implement multi-camera association, there are three hyperparameters to be added to DeepSort:

- **Multi-camera association coefficient** - coefficient that regulates the influence of the feature vector and the absolute position on the final decision.

- **Multi-camera association threshold** - a threshold that justifies that two tracks relate the same person.
- **Multi-camera distance threshold** - the max euclidean distance between two tracks that could be small enough to consider the tracks related to the same person.

Let us define the camera relation matrix. If two tracks are from the same camera, than the entry is 0 and 1 otherwise.

$$A = \begin{array}{c} \begin{matrix} t_1 & t_2 & \dots & t_n \end{matrix} \\ \begin{pmatrix} 0 & 1 & \vdots & 1 \\ 1 & 0 & \vdots & 1 \\ 1 & 0 & \ddots & 1 \\ 1 & 1 & \vdots & 0 \end{pmatrix} \end{array} \begin{matrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{matrix} \tag{5.2}$$

After that, calculate the cosine similarity correlation matrix between tracks feature vectors.

$$B_{i,j} = \cos(\theta) = \frac{F_i F_j}{\|F_i\| \left\|F_j\right\|} \tag{5.3}$$

Then define the distances matrix. Firstly one needs to map relative in-camera position to absolute coordinates (each camera has offset from zero coordinate defined in configuration), then calculate euclidean distance between each track, set to 0 those distances that are greater than *multi-camera distance threshold*, normalize the matrix with min-max normalization to convert entries of the matrix to 0 to 1 range.

$$z = \frac{x - min(x)}{max(x) - min(x)} \tag{5.4}$$

$$C = \begin{array}{c} \begin{matrix} t_1 & t_2 & \dots & t_n \end{matrix} \\ \begin{pmatrix} 0 & 0.8 & \vdots & 0 \\ 0.6 & 0 & \vdots & 0.73 \\ 0.3 & 0.37 & \ddots & 0.95 \\ 0 & 0.87 & \vdots & 0 \end{pmatrix} \end{array} \begin{matrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{matrix} \tag{5.5}$$

The author used weighted some of to combine distance and appearance measurements. If the result exceeds some predefined threshold, than one person could appear on two tracks:

$$R_{i,j} = \begin{cases} x, & \text{if } x > v, \text{ where } x = A_{i,j}(\lambda B_{i,j} + (1 - \lambda) C_{i,j}) \\ 0, & \text{otherwise} \end{cases} \tag{5.6}$$

where:

λ - multi-camera association coefficient $0 \leq \lambda \leq 1$, v - multi-camera association threshold $0 \leq v \leq 1$, A - camera relation coefficient. Entry is 0 if two tracks are from the same camera and 1 otherwise, B - cosine similarity, C - absolute distance coefficient.

Chapter 6

Experiments

Since there is a custom dataset provided by Ricker Lyman Robotic and labeled manually by the author of the paper, all tests will be performed using it as a data source.

6.1 Multi-camera DeepSort

DeepSort framework was designed to work with one camera only and expects the input directly from the object detection model. The author has rewritten the input of Sort part to consume the merged messages from all cameras.

The track entity was enriched by *cam_id* parameter, which is used to distinguish the message by the camera.

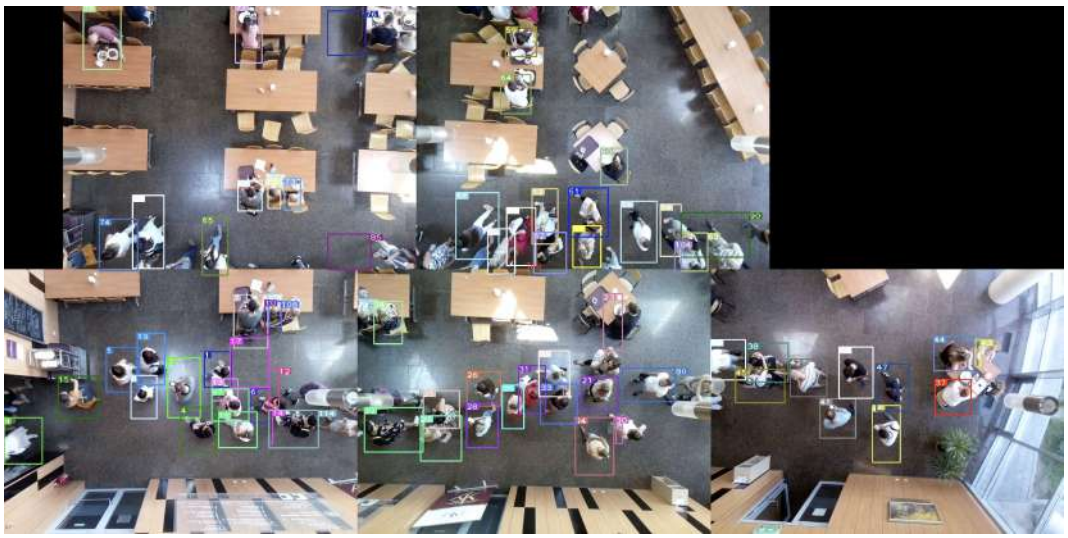

FIGURE 6.1: Result of multi-camera tracking

As a result, one could see that Sort component is agnostic to the number of cameras. It respects the detections from separate cameras and handles them separately.

6.2 Multi-camera id assignment

Sort algorithm finishes its work after performing the tracking phase. But since the author of this thesis added one more step of person re-identification in overlapping cameras, let us consider several cases to see how significant is an impect of *multi-camera association coefficient* for re-identification.

6.2.1 Test 1

Setup: (multi-camera association coefficient=0.3, multi-camera association threshold=0.6, multi-camera distance threshold=100)

Appearance impact: 30% Position impact: 70%

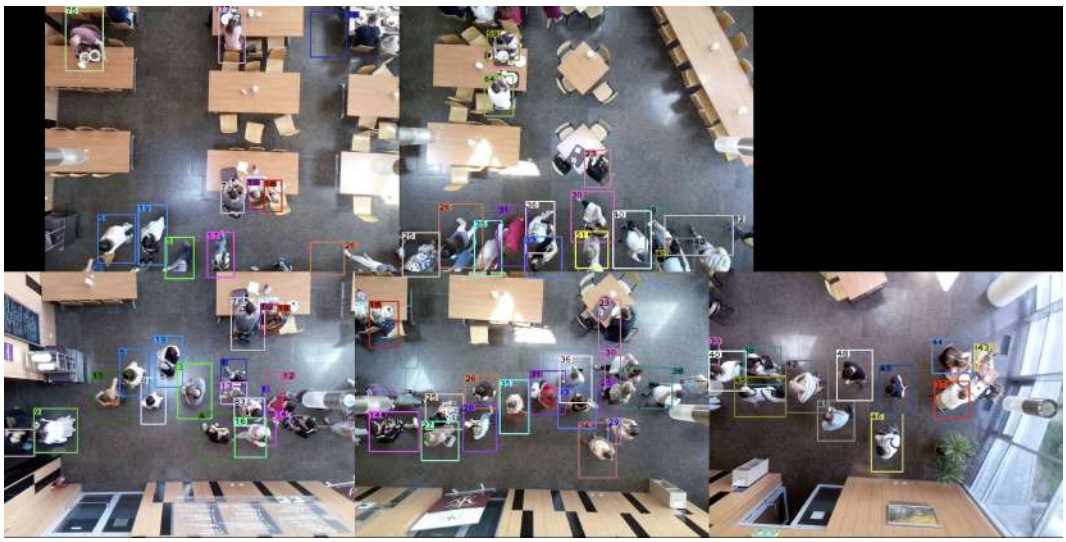

FIGURE 6.2: Test 1. Image 1

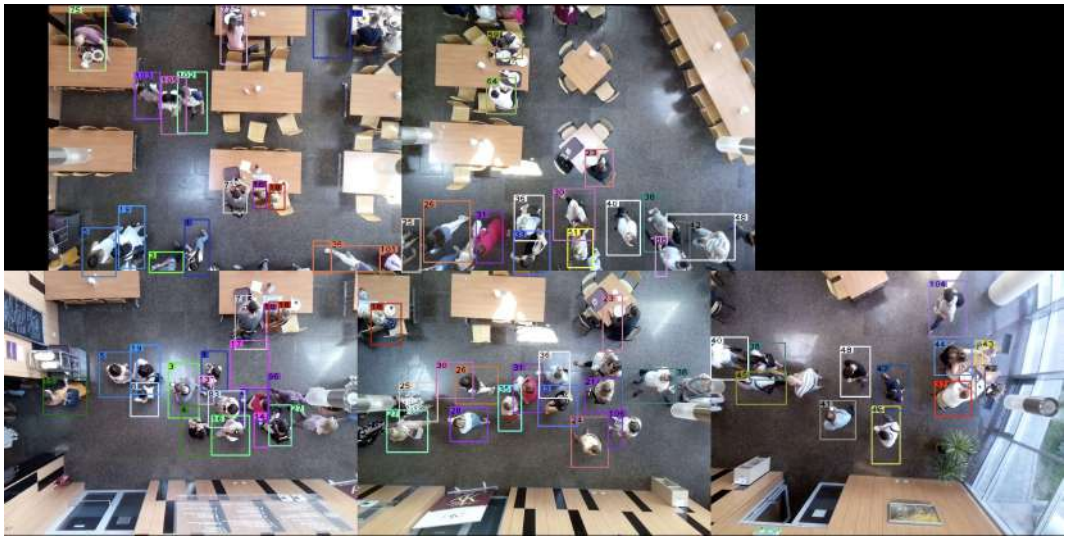

FIGURE 6.3: Test 1. Image 2

6.2.2 Test 2

Setup: (multi-camera association coefficient=0.5, multi-camera association threshold=0.6, multi-camera distance threshold=100)

Appearance impact: 50% Position impact: 50%

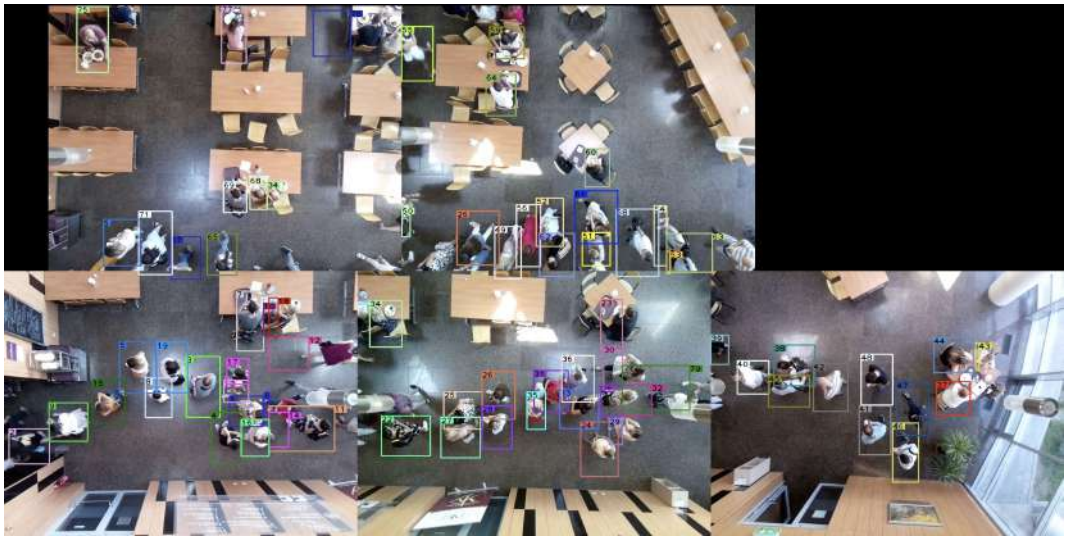

FIGURE 6.4: Test 2. Image 1

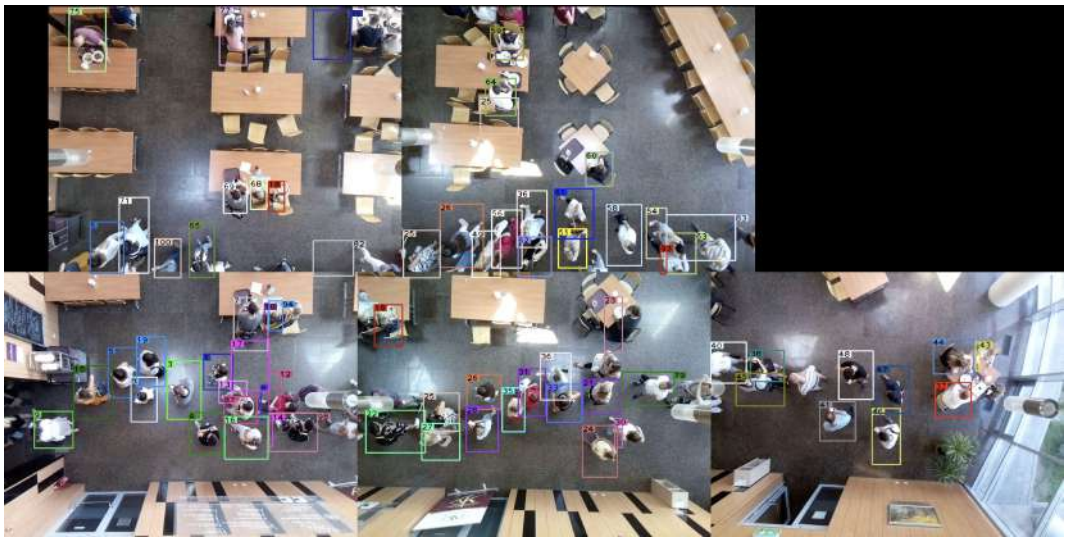

FIGURE 6.5: Test 2. Image 2

6.2.3 Test 3

Setup: (multi-camera association coefficient=0.8, multi-camera association threshold=0.6, multi-camera distance threshold=100)

Appearance impact: 80% Position impact: 20%

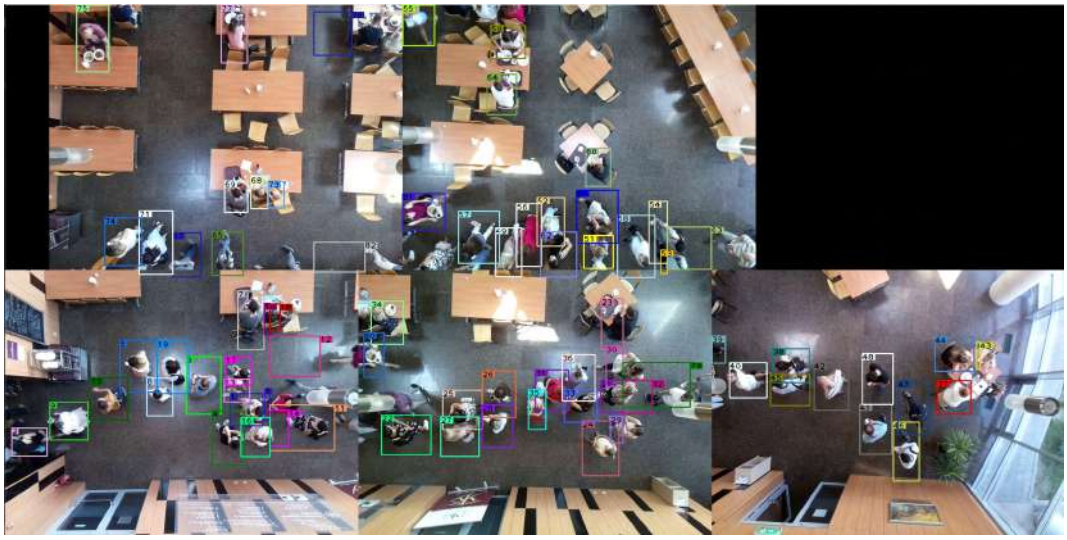

FIGURE 6.6: Test 3. Image 1

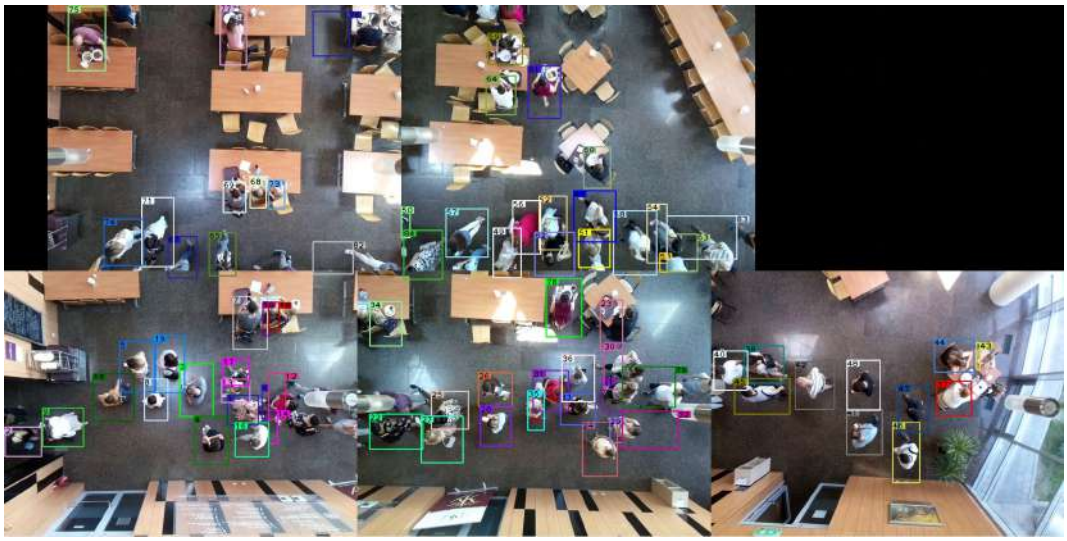

FIGURE 6.7: Test 3. Image 2

As one could see for a specific dataset absolute position of a person is much more important than her appearance. If the dataset consisted of images containing more appearance info, such as the side-view dataset, the impact appearance could have drastically increased.

Chapter 7

Discussion

7.1 Future work

7.1.1 Proposed method improvement

The proposed method of finding the same person on several cameras could be improved by:

- Changing *camera relation coefficient* to a transition matrix, where every row will have non-zero entries for neighboring cameras, thus not compering the tracks that are located on distant cameras.
- Add the statistical model to find the best weights of features using λ multi-camera association coefficient.

7.1.2 Framework for multi-camera person re-identification

During the research, the author has noticed that there is no modular open-source framework for multi-camera object tracking. Despite the existence of many tools solving one or several sub-tasks, no tools generalizing the approach exist. Creating one will significantly reduce efforts of the system setup and maintenance.

7.1.3 Adding data visualization tools

One thing existing solution lacking is integration with visualization tools such as Grafana or Kibana displaying the system health and real-time analytics. This would help in diagnostics and finding useful insights, which in turn generates the business value of the solution.

Chapter 8

Conclusions

The author presented a design of the solution for a top-view tracking system. Besides that, the author proposed an approach to find associate several tracks with the same person.

The system described above is modular, so any part of it could be easily replaced with any of the existing analogs depending on the needs.

Additionally, the author has created a dataset for person detection containing almost 4000 labeled images.

Appendix A

Code

A.1 Pseudocode

```
import numpy as np
from typing import List


camera_offset_from_zero_coordinate = {
    cam_id_1: (x, y),
    cam_id_2: (x, y),
    ...
    cam_id_n: (x, y)
}


def find_possibilities_of_same_ids(features: np.ndarray,
                                   confirmed_tracks_indexes: List,
                                   cam_ids: List
                                   ):
    """
    Defines a cost matrix and checks
    if there are some ids to associate them
    """
    # find cosine similarity
    cs_cost_matrix = calculate_cosine_similarity(features, features)

    # calculate the central point of each track
    central_points = [
        get_track(i).get_central_point() for i in
    confirmed_tracks_indexes
    ]

    # convert camera relative positions to absolute positions
    for i, track_index in enumerate(confirmed_tracks_indexes):
        cam_id = cam_ids[i]
        x, y = central_points[i]
        # find the offset of each camera by cam_id
        offset_x, offset_y = camera_offset_from_zero_coordinate[cam_id]
        central_points[i] = np.array([offset_x + x, offset_y + y])

    # calculate euclidean distance between absolute positioned points
    distances = np.zeros(cs_cost_matrix.shape)
    for i, p1 in enumerate(central_points):
        for j, p2 in enumerate(central_points):
            distances[i, j] = np.linalg.norm(p1 - p2)

    # set those entries that are higher than threshold to 0
    distances[distances > multi_camera_maximum_distance] = 0

```

```
    # apply min-max normalization to convert distances to (0 to 1)
    scale
    distances =
        ( distances - distances.min() ) /
        ( distances.min() + distances.max() ),

    # after normalization distances that are close to each other
    # are close to 0, while those that are relatively distant are close
     to 1
    # change this it vice-versa
    distances = np.where(distances == 0, 0, 1 - distances)

    # calculate relation to the same camera
    cam_coefficients = np.ones(cs_cost_matrix.shape)
    for i in cam_ids:
        for j in cam_ids:
            # if tracks have same_cam ids they are definitely different
            # set cam coefficient to zero
            if i == j:
                cam_coefficients[i, j] = 0


    # compute the probability of setting same ids
    result = cam_coefficients * \
        np.add(
            multi_camera_association_coefficient * cs_cost_matrix,
            (1 - multi_camera_association_coefficient) * distances,
        )
    return result


def replace_ids(probs: np.ndarray):
    # all values that are less than threshold set to 0
    probs[probs < multi_camera_association_threshold] = 0
    max_val_indexes = np.argmax(probs, axis=0)
    replace_ids_of_tracks_that_left(max_val_indexes)



# start reading here!
def update_framework_step(self,
                          features: np.ndarray,
                          confirmed_tracks_indexes: List[int],
                          cam_ids: List[int]):
    """
    :param self: Tracker
    :param features: feature vectors of currently existing tracks
    :param confirmed_tracks_indexes: indexes of confirmed tracks only
    :param cam_ids: cam_ids of tracks
    :return:
    """

    # do usual Sort update

    # try to find similar tracks and assign same ids to them
    if self.frame_index > self.n_init and len(features) > 0:
        # method below returns probability matrix
        # of two tracks tracking the same person
        probabilities = find_probs_of_same_ids(
            features,
            targets,
            confirmed_tracks_indexes,
            cam_ids
```

```
        )
        # replace ids of the
        replace_ids(probabilities)
```

CODE SAMPLE A.1: Pseudo code of re-identification in overlapping multi-camera system

A.2 Message schemas

```
@namespace("ucu.master.diploma.motility")
protocol Motility {

  record Detection {
    float confidence;
    array<float> feature = [];
    array<float> tlwh = [];
  }

  record Msg {
    int cam_id;
    timestamp_ms event_time;
    array<Detection> detections = [];
  }

}
```

CODE SAMPLE A.2: Message (avro)schema

Bibliography

[1] Lucas Beyer Alexander Hermans and Bastian Leibe. "In Defense of the Triplet Loss for Person Re-Identification". In: *arXiv arXiv:1703.07737v4* (2017).

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] John K. Zao Chuck Byers Ron Zahavi. *The Edge Computing Advantage*. Tech. rep. Version 1.0. An Industrial Internet Consortium White Paper, 2019.

[4] Nir Ailon Elad Hoffer. "Deep Metric Learning Using Triplet Network". In: *arXiv arXiv:1412.6622v4* (2017).

[5] Ross Girshick Ali Farhadi Joseph Redmon Santosh Divvala. "You Only Look Once: Unified, Real-Time Object Detection". In: *arXiv arXiv:1506.02640* (2016).

[6] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[7] Martin Kleppmann. *Designing Data-Intensive Applications*. O'Reilly Media, Inc., 2017.

[8] Yi Yang Liang Zheng and Alexander G. Hauptmann. "Person Re-identification: Past, Present and Future". In: *arXiv:1610.02984v1* (2015).

[9] Kot ACC Lu Z Jiang X. "Deep coupled ResNet for low-resolution face recognition". In: *IEEE Signal Processing Letters* 25 (2018), pp. 526–530.

[10] Gwen Shapira Neha Narkhede and Todd Palino. *Kafka: The Definitive Guide*. O'Reilly Media, Inc., 2017.

[11] Dietrich Paulus Nicolai Wojke Alex Bewley. "Simple Online and Realtime Tracking with a Deep Association Metric". In: *arXiv arXiv:1703.07402* (2017).

[12] Trevor Darrell Jitendra Malik Ross Girshick Jeff Donahue. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *arXiv arXiv:1311.2524* (2014).

[13] Riccardo Satta. "Appearance Descriptors for Person Re-identification: a Comprehensive Review". In: *arXiv arXiv:1307.5748* (2013).

[14] Ellen Friedman Ted Dunning. *Streaming Architecture*. O'Reilly Media, Inc., 2016.

[15] Zhihao Zhou Xiying Li. "Object Re-Identification Based on Deep Learning". In: *intechopen 10.5772/intechopen.86564* (2019).

[16] Kaiyang Zhou and Tao Xiang. *Torchreid: A Library for Deep Learning Person Re-Identification in Pytorch*. Tech. rep. 2019.

[17] Kaiyang Zhou et al. "Learning Generalisable Omni-Scale Representations for Person Re-Identification". In: *arXiv preprint arXiv:1910.06827* (2019).

[18] Kaiyang Zhou et al. "Omni-Scale Feature Learning for Person Re-Identification". In: *ICCV*. 2019.