VUT FIT POVa

Pedestrian Tracking

Lukáš Petrovič, Filip Šťastný, Martin Vondráček,
{xpetro11,xstast24,xvondr20}@stud.fit.vutbr.cz

December 28, 2018

1 Introduction

The aim of this project was to create a computer vision system capable of tracking pedestrians in video sequences captured by multiple stationary cameras. The input should be video streams from the cameras place in a room. The output should be a visualisation of tracks of the people who appeared in front of the cameras. A flow of the system can be described by the following steps:

1. Let's have a room with 1 or multiple cameras with known positions.
2. Record videos of people walking in the room.
3. Detect figures in the videos.
4. Find out matching figures (the same person) across the videos from multiple cameras.
5. Locate the person in 3D space.
6. Track the targets across the video sequence (chronologically).
7. Mark down 2D track for each person and visualise it in a floor plan of the room.

2 Computer Vision Research

This section briefly summarises our study in the field of computer vision. We have analysed relevant research focused on human detection, similar image matching, triangulation, and pedestrian tracking. We have also searched for existing solutions, available datasets, and other relevant information sources.

2.1 Human Detection

Multiple options for human detection exist. We could mention *frame differencing* that can easily detect moving objects. The method computes the absolute difference between the previous frame and current frame. The detected moving objects are identified as humans by analysing the blobs, and their dimensions [CR15].

Another technique is called *Histogram of Oriented Gradients* (HoG). HoG is an object detection technique, which uses distribution of intensity gradients or edge directions to define the shape of an object. It decomposes an image into cells, and for each cell, histogram of gradients is computed. Filtering is used to find the gradients in both horizontal and vertical directions. SVM or Adaboost are then used for identifying and detecting humans from obtained features [CR15].

We used the modern solution called *OpenPose*[1] to detect human poses in images. We choose this multi-person keypoint detection library because it is reliable and has better results then previous mentioned techniques. It is also time invariant to number of detected people. It uses deep neural network to obtain features like joints and connections between them. We used pretrained model, which had been trained on the COCO dataset[2]. Several resources concerning intergration of *OpenPose* are available online[3] [4].

[1] https://github.com/CMU-Perceptual-Computing-Lab/openpose

[2] http://cocodataset.org/

[3] https://www.learnopencv.com/deep-learning-based-human-pose-estimation-using-opencv-cpp-python/

[4] https://www.learnopencv.com/multi-person-pose-estimation-in-opencv-using-openpose/

2.2 Similar Image Matching

Similar image matching is used to match the same person captured by multiple cameras at a time. We got multiple images of the person from multiple cameras, and we need to verify if it is the same person or not.

For this purpose we can use algorithms to extract image features (Harris Corner detector, etc.) and try to find them in another image. However, if the image is scaled (multiple cameras = multiple distances and angles), the features may become obsolete (corner in a small image may not be a corner in a large one). This problem can be solved by scale invariant features, such as are used by SIFT and SURF algorithms [5].

In this project, we used histogram based approach. Image histogram is a representation of colour distribution in the image (it sums number of pixels for each colour). Histogram comparison in OpenCV can be done using multiple methods[6]: intersection, correlation, chi-square or Bhattacharyya. Normalised histograms are quite tolerant in a meaning of viewing angle or image size.

2.3 Triangulation

By the term *triangulation* we refer to the process of determining 3D location of an observed subject from two its perspective projections. Triangulation can be used not only to locate a subject, bot also for 3D reconstruction of subject's model [7]. One of the possible approaches utilises pinhole camera model. In such a case, 3D parts of an observed subject are projected into 2D images using perspective transformation [8]. Based this perspective transformation, it is possible to derive transformation from 2D image to 3D coordinate system in front of the camera. However, calibration is required in order to obtain focal length of each used camera.

Camera's calibration — measurement of its focal length can be done based on perspective projection of a known subject in a known distance. This approach is based on triangle similarity, which is also visible in Figure 1.

$$f = \frac{z_{projected} \cdot x_{real}}{z_{real}} \tag{1}$$

[5] https://docs.opencv.org/3.4/db/d27/tutorial_py_table_of_contents_feature2d.html
[6] https://docs.opencv.org/3.4/d8/dc8/tutorial_histogram_comparison.html
[7] https://www.csie.ntu.edu.tw/ cyy/courses/vfx/16spring/lectures/handouts/SFMedu_brief.pdf
[8] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

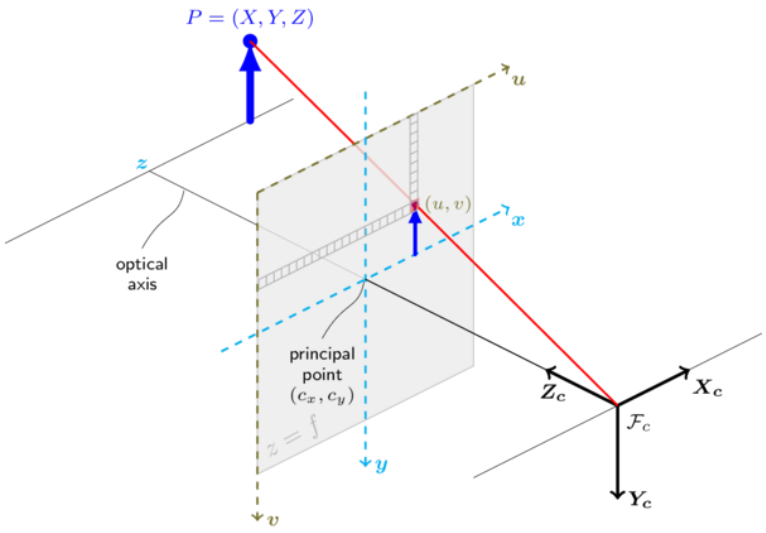

Figure 1: Pinhole camera model, taken from OpenCV API Reference [Tea18]

Focal length is computed according to Equation 1, where x_{real} is known real distance of a subject from camera, z_{real} is known real height of a subject, $z_{projected}$ is height of a projected 2D image of subject measured in pixels, and f is computed focal length. It is important to note that computed focal length of a given camera does not change in different scenes as long as camera's settings like zoom stay the same.

It is possible to calculate subject's distance from a single camera based on subject's real world size and camera's focal length [9] [10] [11]. Equation 2 shows computation of real distance from camera, based on Equation 1.

$$x_{real} = \frac{z_{real} \cdot f}{z_{projected}} \tag{2}$$

[9] https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv

[10] https://www.scantips.com/lights/subjectdistance.html

[11] https://www.youtube.com/watch?v=sW4CVI51jDY

2.4 Pedestrian Tracking

Most methods suitable for pedestrian tracking consist of two parts: pedestrian detection and then tracking based on similar features of detected areas. Methods outlined in subsection 2.1 can be utilised for pedestrian detection. Similarly, subsection 2.2 offers way how to detect which detected areas are similar. Analysed solutions include background model with foreground separation, probability and histogram tracking of foreground objects, and histograms of oriented gradients with neural networks [ST11] [AT17].

2.5 Datasets

From searched and analysed datasets, we consider COCO dataset[12] a feasible one for training of human detectors (subsection 2.1). Other useful datasets containing pedestrians are MPII Human Pose Dataset [APGS14] and Human Pose Evaluator Dataset [JZE+12].

3 Implemented solution

This section outlines key topics of implemented pedestrian tracking application. Our solution was designed according to studied researches and methods mentioned in section 2. Further implementation details are available in a form on source code documentation.

3.1 Software Architecture

The task of pedestrian tracking can be divided into several sub-tasks: video processing, people detection in 2D, person view matching, person triangulation (location), tracking of located people over time. These sub-tasks correspond to individual components with specified input and output interfaces. Upon connecting all mentioned components together, a processing pipeline is established. Diagram of the processing pipeline is shown in Figure 2. The pipeline is implemented against abstract base classes for each used component, this allowed flexible parallel development of different components which can be easily interchanged.

[12] http://cocodataset.org/

Each observed scene is characterised by at least two configured cameras (front, side) at static positions with known orientations. Instance of any class inherited from abstract base class `ImageProvider` is responsible for sequentially providing 2D images from adequate data source. One of the implemented providers is `ImageProviderFromVideo`, which can load images from video files.

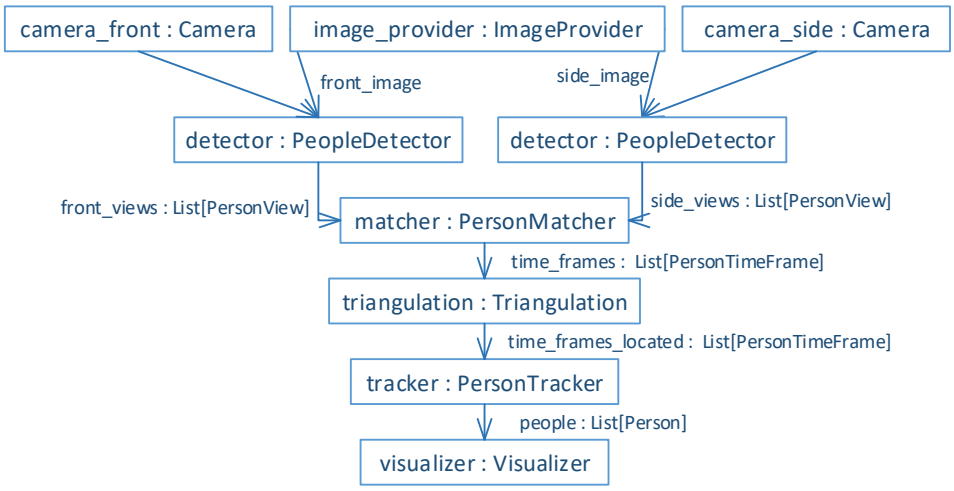

Figure 2: Implemented processing pipeline

3.2 People Detection in 2D

Based on our study of human detection methods, which was briefly outlined in subsection 2.1, our team has decided to implement 2D person detector based on *OpenPose* deep neural network. We are using official model pretrained on COCO dataset (*pose_iter_440000.caffemodel*), as mentioned earlier.

Another possible alternative is BODY_25 model (*pose_iter_584000.caffemodel*) pretrained also on COCO dataset, but with additional body parts, especially for hands. The BODY_25 model should be about 40% faster on GPU, but nearly 3x faster on CPU. Since just some of computers has a GPU supporting *opencl-1.2* and *cuDNN*, we picked COCO model as default.

Our solution evaluates only joint connections that are relevant for our further processing. These include left hip, right hip, and a neck. Other than these, we created bounding-boxes for each detected person. These bounding-boxes are used to pair detected person in views from both cameras.

3.3 Person View Matching

PersonView contains information about a detected person - image, bounding box, coordinates. We use histogram based matching to link person views from multiple cameras at a time, and create a *PersonTimeFrame*, which contains all views matched to the corresponding single person and a timestamp.

From other methods mentioned in subsection 2.2, SURF algorithm could provide a good solution, but it is not supported by *OpenCV 3+* anymore (because of legal issues) and it is packed in a complementary package, which could be problematic to get working on some systems.

We calculate histograms for any person's views from all cameras. Then the histograms are compared using two methods:

1. Intersection method - higher result value means better match.
2. Bhattacharyya (Hellinger) method - lower result value means better match.

The views are compared and person confirmed only if both methods select the same views as the best matching ones from all other views. The 2-method confirmation is done because of the comparison results are too variable and a threshold can not be set properly to filter any useful amount of matches.

3.4 Triangulation

Triangulation of detected and matched people is implemented as component `CameraDistanceTriangulation`, which implements interface defined by `Triangulation`. This component solves detection of subject's 3D position according to subsection 2.3.

Used cameras were previously calibrated (according to Equation 1), so their focal lengths are known. Considering height of average person's torso, it is possible to calculate person's distance from camera using Equation 2. This process is applied to images from all available views of the given person, which produces distance planes in 3D space. These planes represent possible

distance from each camera. Observed person is positioned on these planes. Intersection of two planes produces a straight line of person's position. The final step is then performed by selecting a single point on such line at a selected height (z axis). Whole process is illustrated in Figure 3, where the subject is located at the same height (z axis) where cameras are positioned.

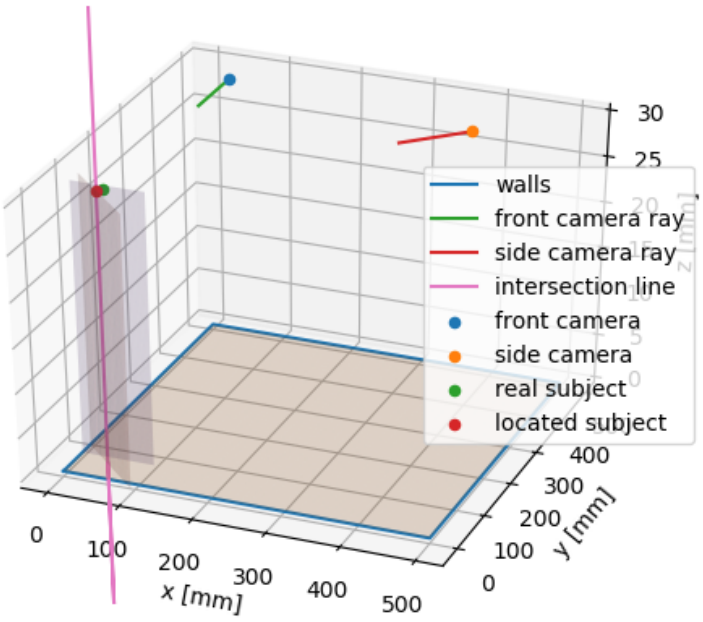

Figure 3: Triangulation using distance from two cameras

3.5 Tracking

Tracking of detected and located people moving in the observed scene over time is realised by `HistogramTracker` with interface of `PersonTracker`. This component takes single `PersonTimeFrame` and decides whether detected person have been previously seen and is already being tracked or whether it is a new person.

Tracker keeps evidence of all tracked people and all their `PersonTimeFrames`.

Output from tracking a frame is a `Person` to which current frame belongs. Tracking utilises histograms to decide whether a detected frame is similar to frames of any already tracked person. This was already described in subsection 2.2 and subsection 2.4, histograms are compared using *Bhattacharyya distance* and *Intersection method*.

3.6 Visualisation

When pedestrians' 3D positions and paths are successfully tracked, it is important to visualise processed information to the user in a easy to use and understandable way. Component `Plotter3D` with interface of `Visualizer` has such responsibility. It is initialised with static information about observed scene and a connection for retrieving updated list of tracked people.

Plotter creates separate window and renders a model of current 3D space (x, y, z axes). This visualisation shows position and orientation of cameras and path (connected positions) of all tracked people. Individual people are distinguished by their unique *ID* and their paths have different colours. User is free to use mouse to rotate and zoom 3D model. Example visualisation is shown in Figure 4.

Because the pedestrian tracking process is a computationally demanding cyclic operation, visualisation is performed in a separate thread. This allows user to interact with the 3D model while the pipeline is processing next set of input images.

4 Evaluation

Following text summarises testing of most important parts of the implemented solution. We have analysed speed and precision of individual components. We have captured and used total of 22 own images from 3D scenes with walking people and 4 own videos from these scenes. Files are identified by scene number (prefix `s1`, `s2`, and `s3`), camera identification and position (`f` and `front`, `m` and `side`), number of people (`single`, `multi`). Therefore, our testing data consist of following files which are further described in `testing/_data/README.md`.

```
testing_data/s1_front_d150_h50.jpg
testing_data/s1_front_d400.jpg
testing_data/s1_side_d500.jpg
```

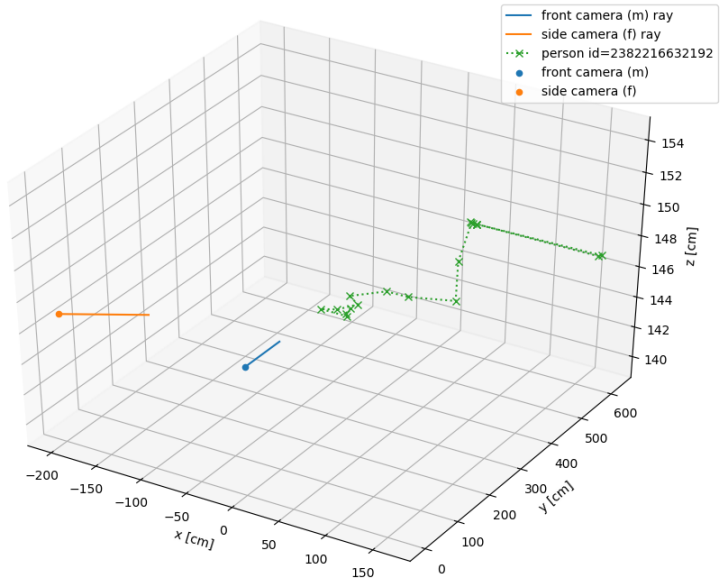

Figure 4: Visualisation with implemented `Plotter3D`.

```
testing_data/s2_f_x0y300.png
testing_data/s2_f_x0y600.png
testing_data/s2_m_x0y300.png
testing_data/s2_m_x0y600.png
testing_data/s3_f_side_multi_y600.png
testing_data/s3_f_side_single_F_y600.png
testing_data/s3_f_side_single_x0y300.png
testing_data/s3_f_side_single_x50y600.png
testing_data/s3_m_front_multi_bg.png
testing_data/s3_m_front_multi_y600.png
testing_data/s3_m_front_single_F_y600.png
testing_data/s3_m_front_single_x0y300.png
testing_data/s3_m_front_single_x50y600.png
testing_data/s3_single_30_front.png
testing_data/s3_single_30_side.png
```

```
testing_data/s3_single_49_front.png
testing_data/s3_single_49_side.png
testing_data/s3_single_51_front.png
testing_data/s3_single_51_side.png
testing_data/s3_f_side_multi.mov
testing_data/s3_f_side_single.mov
testing_data/s3_m_front_multi.mov
testing_data/s3_m_front_single.mov
```

4.1 People Detection in 2D – OpenPose

Implementation of people detector based on OpenPose network was evaluated both for speed and precision. Detector is able to identify key parts of human body with precision of 10 pixels in testing images, these tests are implemented in `test_povaPose.py` unit tests. On the other hand, detection process was not very fast. Figure 5 shows speed evaluation for different computers.

4.2 Triangulation

Implementation of `CameraDistanceTriangulation` was tested and evaluated using unit tests located in `test_triangulation.py`. Achieved precision of determining location was within 10 cm delta, in some cases 15 cm delta, and in worst case within 35 cm delta in scene where people were walking in 6 m space.

5 Conclusion

Contributions:

- Lukáš Petrovič: person detection based on OpenPose
- Filip Šťastný: person matching, image preprocessing, video preprocessing, testing data
- Martin Vondráček: triangulation, person tracking, overall architecture, visualisation

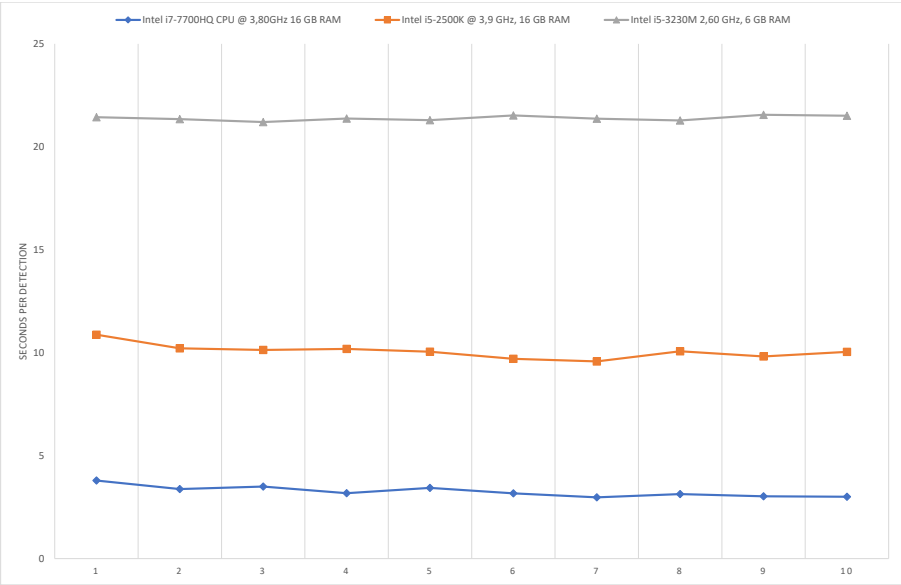

Figure 5: Speed evaluation for repeated detection of person in testing image.

During this project we have implemented a computer vision system intended for tracking pedestrians in observed scene. We have employed multiple cameras. Our system is capable of detecting people in images from two cameras. Detected bodies are then matched together based on similarities in their histograms in order to make a pair of images of the same person. Detected person is then located in 3D space using triangulation. Triangulation uses depth planes in 3D space and their intersection. Located frames of people are then tracked to form path in space over time.

Our testing and evaluation confirmed that person detection based on OpenPose can gain precise results. However, this process is computationally demanding and slow. Evaluation of Triangulation proved this approach to be sufficient for this application. Unfortunately, our triangulation method suffers from the fact, that OpenPose sometimes detects neck and hips on slightly different positions. This causes fluctuations in determining the right position in 3D space. Another thing that could be improved is the tracker.

At this moment, implemented tracking procedure can track one person successfully. Unfortunately, tracking multiple people in one scene with similar histograms proved to be more difficult to implement.

References

[APGS14] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[AT17] Md Zahangir Alom and Tarek M Taha. Robust multi-view pedestrian tracking using neural networks. In *Aerospace and Electronics Conference (NAECON), 2017 IEEE National*, pages 17–22. IEEE, 2017.

[CR15] Chithira S Vidhya Balasubramanian Chakravartula Raghavachari, Aparna V. A comparative study of vision based human detection techniques in people counting applications. In *Procedia Computer Science*, volume 58, pages 461–469. Elsevier, 2015.

[JZE+12] N. Jammalamadaka, A. Zisserman, M. Eichner, V. Ferrari, and C. V. Jawahar. Has my algorithm succeeded? an evaluator for human pose estimators. In *European Conference on Computer Vision*, 2012.

[ST11] Saman Saadat and Kardi Teknomo. Automation of pedestrian tracking in a crowded situation. In *Pedestrian and Evacuation Dynamics*, pages 231–239. Springer, 2011.

[Tea18] OpenCV Dev Team. Camera calibration and 3d reconstruction, opencv api reference, 2018. [Online; accessed 28-December-2018].