TEK5030 Social Distance Estimation

Birk Sebastian Torpmann-Hagen, Ludvig Løken Sundøen, Nadia Garson Wangberg

October 8, 2021

1 Abstract

In this project we designed and implemented a system which detects whether people are upholding the current social distancing rules. We used stereo binary SGBM to calculate depth, a Convolutional Neural Network (CNN) to detect people. By using both the dense stereo depth together with the detected people, the position of people in relation to the camera $\mathbf{t}_{pers}^{cam}$ was found found in 3D. The distance between two people could be found by finding the euclidean distance between vectors. This can be used to see if people are upholding the current social distancing rules. For data we used ETH Zurich's Moving Obstacle Detection from Mobile Platforms dataset [2].

Figure 1: Close up of final result

2 Person detection

We employed a deep-learning based approach for detecting persons. Since the system should (given sufficient development) be more or less capable of running in real time, a quite lightweight network is necessary. To this end, we used a pretrained YOLOv3 derivative known as Darknet [5]. On proper hardware, Darknet is capable of running at 30 FPS.

2.1 Model adjustment

Since the pretrained model is designed to detect more classes of objects than just persons, the network head was adjusted such that it instead only returned bounding boxes for people. Moreover, since we are less interested in the bounding box than just getting a rough position estimate, the resulting bounding box outputs were post-processed, returning the center coordinate.

2.2 Using the Darknet model

Originally, we planned to employ a skeleton-based detection method, which in theory is more robust to occlusions, of which there are many in our dataset. However during preliminary analysis such an approach was shown to exhibit much higher execution time and evidently a high number of false positives. As such, we decided that our Darknet model was sufficient.

Finally, preliminary experiments on the Coco datset showed that the adapted Darknet had a large number of false negatives - i.e where the model failed to identify persons at all. This is likely due to the fact that the recommended confidence thresholds given in the literature are selected to increase precision across a large number of classes. Since, however, we only required the one class - person - we could reduce this confidence threshold. This increased the sensitivity of the model to persons further from the camera. The model still sometimes fails, however, but this is typically due to heavy occlusion. Examples have been provided in the appendix.

$$(u, v) = personDetector(img_{left}) \quad (1)$$

3 Dense stereo image processing

In this project a dense depth map was found from the SGBM algorithm. This was based on this course's lab7. The resulting depth image can be seen in figure 2

3.1 Stereo calibration

In order to use stereo data to generate a depth images, rectified images are needed. Stereo calibration outputs both intrinsic parameters K as shown below, as well as extrinsic parameters $[R, t]$ representing the transformation between the two cameras.

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

In this project the camera intrinsic and extrinsic parameters were given from ETH Zurich dataset, but algorithms like Zhang's method could also be used to find the camera parameters.

3.2 Depth from stereo binary SGBM

Stereo binary SGBM was used to calculate the disparity image from the rectified stereo image. The disparity image was used to find the dense depth image as shown in the equations below. In the equation f is the focal length and d is the depth image.

$$disparity = SGBM(img_{left}, img_{right}) \quad (3a)$$

$$d = f\frac{baseline}{disparity} \quad (3b)$$

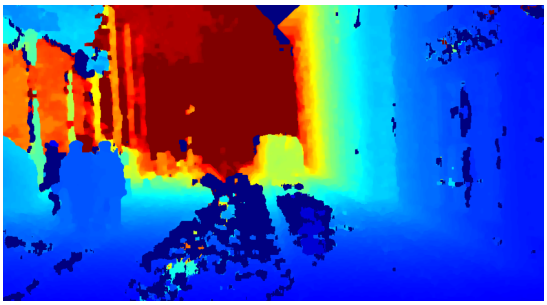

Figure 2: Depth image produced from SGBM

4 Person 3D placer

$$t^{cam}_{pers} = person3Dplacer(d, [u, v], K, [R, t]) \quad (4)$$

The purpose of the person placer algorithm is to find the 3D translation between the camera and the detected person. The input of this algorithm are both the detected person (u,v) and the depth image (d). As well as the intrinsics of the camera. As explained above the CNN detects the person in pixel coordinates (u,v), while the depth image is from the Stereo Binary SGBM algorithm. The output of the person placer algorithm is on the form as in equation 5.

$$t_{pers}^{cam} = \begin{bmatrix} x_{pers}^{cam} \\ y_{pers}^{cam} \\ z_{pers}^{cam} \end{bmatrix} \tag{5}$$

4.1 A Naive Solution

Our initial idea was to use the width w and height h of the image in pixels, field of view of the camera FOV_u and FOV_v, depth image ($depth$) and detections (u, v) to estimate the t_{pers}^{cam} vector. The solution was based on simple trigonometry, but made several assumptions. If the person was detected in the middle of the image the vector would be $\mathbf{x}_{pers}^{cam} = [0, 0, depth(w/2, h/2)]$. This algorithm would be simple and naive. One of its assumptions was that the center of the pixel image was equal to the projected image center. With other words the intrinsic parameters c_x and c_y were not taken into account. Our initial attempt also did not use focal length, which would not produce the correct results.

4.2 Solution based on pointcloud generation

We quickly realized that what we attempted to achieve was essentially converting a depth image to a 3D point-cloud. This can be done using the stereo calibration parameters and the pinhole camera model. One implementation can be found on Medium [3]. This algorithm use the all the intrinsic parameters and can therefor achieve a higher accuracy. The resulting equation is based on the pinhole camera model equation 3. First x and y over z must be found from $\frac{x}{z} = \frac{c_x - u}{f_x}$ and $\frac{y}{z} = \frac{c_y - v}{f_y}$. From this the resulting 3d vector can be found as shown in equation 6.

$$\mathbf{t}_{pers}^{cam} = \begin{bmatrix} z\frac{x}{z} \\ z\frac{y}{z} \\ \frac{d}{\sqrt{1 + z\frac{x}{z}^2 + z\frac{y}{z}^2}} \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} u \\ v \\ 1 \\ 1/z \end{bmatrix} = \frac{1}{z} \underbrace{\begin{bmatrix} \boldsymbol{K} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}}_{4\times4} \underbrace{\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix}}_{4\times4} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Figure 3: projection of 3d points to 2d pixel coordinates

5 Social distance estimation

From the camera to person translations t_{pers}^{cam}, the distance each person has to the closest person can be found. This is done by finding the euclidean distance from each person to every other person and choosing the minimum distance. This of course leads to $O(n^2)$ where n is the number of persons. This social distance is thresholded in order to decide whether the current social distancing rules are upheld. Each person is assigned the color green if they are further away than 2 meters, yellow if between 1 and 2 meters and red if less than 1 meter.

$$socialdist_i = \min_{j \subseteq 1 \ldots n} \left\| t_{pers,i}^{cam} - t_{pers,j}^{cam} \right\|$$

$$i \neq j$$

6 Software integration and parallelism

As of now this project does not support live processing. This means that the CNN python script is run separately, producing .json files with the detection data of the dataset. Afterwards a C++ program imports both the json and image data, does dense stereo processing and estimates the distance between the people and checks whether the social distancing rules are upheld.

6.1 ROS for parallel programming

In order for our project to be used in real time ROS could be a good solution. ROS is an open source robotics middle-ware, which allows several processes to run in parallel communicating in a message passing framework. ROS is also compatible with both C++ and Python allowing a seamless integration between the two programming languages. We wanted to integrate our project with a SLAM solution, allowing all people detected to be placed in world frame. As we had limited time, this integration had to be postponed. How this could be done is explained in section 8. It is important to note that communication protocols such as Mqtt may enable live processing just as well as ROS.

In this project we have created a few ROS nodes communicating with each other and performing some of the functionality needed in order to achieve live processing. Most of the nodes however are currently empty, and our project is currently not fully implemented in the ROS framework.

6.2 Converting Software to ROS nodes

Below is a figure showing all the ROS nodes and how they communicate together. The circles represents ROS nodes (processes), while the rectangles represent ROS topics, where messages are sent and received. RTABmap has not been implemented by us, it is a popular algorithm for performing real time stereo SLAM [4])

Our main struggle was working with images in ROS, specifically converting between ROS image types and openCV images types. This wasn't a complete blocker, but was definitely more tedious than anticipated. We therefor decided to focus on implementing the project without ROS and without live processing.

Below is an explanation of the tasks of each ROS node. Again it is important to note that this table shows how the nodes are intended to work. The resulting rqt-graph is also attached in Figure 9 in the Appendix.

- **Person detector (Section 2)** : runs the CNN and publishes the pixel position of the person (u,v)
- **Stereo image folder publisher (Section 3)**: Publishes a dataset from folder onto the ROS network as rectified left and right images. Runs the Stereo Binary SGBM algorithm and outputs the dense depth image.
- **Object placer (Section 4)**: Inputs both the dense depth image and the detected person (u,v), output the vector between the camera and the person
- **Stereo RTAB-map (Section 8.1)**: Runs SLAM, publishes a ROS tf-transform between the world frame and the camera frame.

7 Results

The final result can be seen in figure 1 and figure 5. Which shows that the two front most people are not upholding the current social distancing rules.

7.1 Top down plot of people

In Figure 1, you can see by the red color that the two persons are not upholding social distancing rules. Figure 6 is Geogebra plot showing the top down view of the people in Figure 1. This plot was created to validate whether the results seemed plausible. It plots $\mathbf{t}^{cam}_{pers,left}$ and $\mathbf{t}^{cam}_{pers,right}$ in the zx-plane. Origo in this plot is the pinhole of the left camera. The distance between the vectors are 0.52 as shown in this plot as well as Figure 1. It would be interesting

to create such a plot for every frame, resulting in a movement trajectory for each person. Use-cases for this is discussed in section 8.3.

Figure 4: Result of person detection

Figure 5: Final result

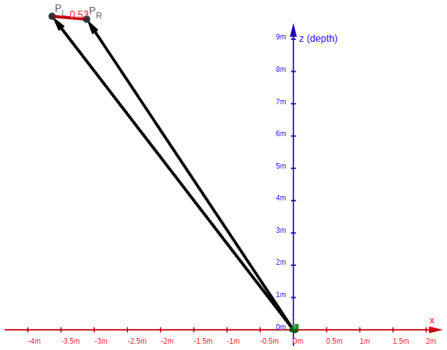

Figure 6: Top-down plot of the two front most people in figure 5. Origo is the pinhole of the left camera

7.2 Evaluation of distance estimation

As we do not have ground truth data on the actual social distance it is difficult to evaluate the exact accuracy of our algorithm. The distance found is however visually looking alright, which should be good enough for this use case. As can be seen in figure 1, the detections are slightly offset to the left. This is not due to the CNN detection algorithm itself, but rather due to it running on the un-rectified images instead of the rectified images. This could be solved by rectifying before running detection.

7.3 Evaluation of person detection

The pretrained detection model seemed to generalize sufficiently well to the stereo data set. As mentioned previously however, this bounding-box based approach appears fairly weak to occlusion. Naturally, this is of some concern considering the fact that major violations of the social distancing guidelines typically result in some level of occlusion.

8 Further works

8.1 integration with SLAM

By integrating this project with SLAM or Visual odometry a people map could be produced with a moving camera. The use case of this could be mapping where huge crowds of people is located. Our current project find the translation between people and the camera t^{cam}_{pers}. By assuming the rotation between the camera and people doesn't matter, the SE(3) transformation matrix would be

$$T^{cam}_{pers} = \begin{bmatrix} \mathbb{I} & t^{cam}_{pers} \\ 0 & 1 \end{bmatrix}.$$

The output of SLAM is the pose of the camera in relation to some fixed world position,

this can be given as an SE(3) transformation matrix

$$T_{camera}^{world} = \begin{bmatrix} R_{camera}^{world} & t_{camera}^{world} \\ 0 & 1 \end{bmatrix}.$$

The final people map could be found by $T_{pers}^{world} = T_{cam}^{world} T_{pers}^{cam}$.

8.2 Detection model reduction

Given sufficient computational resources and some dataset processing, we could likely effectivize the detection component of our system by orders of magnitude by training a smaller network from scratch. As mentioned earlier, the model we are using is pretrained on the Coco dataset, and as a consequence encodes feature mappings that identify more classes than just "person". Since only a few of the parameters in the model actually detect instances of people, it stands to reason that a model with just these parameters would achieve equivalent performance, but be much more efficient.

8.3 Tracking through time

Prior to starting this project, we thought it would be interesting to attempt to track those who violate the social distancing guidelines through time. This could for instance be used to assign a risk factor to the individuals, which (although a bit 1984-esque) may have been useful for contagion control and perhaps contact tracing. Consider for example when venues such as restaurants or movie theatres or the like start to open again; cameras could for example track individuals during their stay, and if infection ever occured, it would be fairly simple to use a system such as ours to assign infection probabilities to every attending member since accurate distance and exposure time data would be available. The algorithms required to do such tracking through time have already been developed and well tested, and as such it would only be a practice in integration to get it working for our system as well. We planned to use SORT [1], which combines a Kalman filter and the Hungarian method to establish temporal correspondences. This is in contrast to competing appraoches which often instead relies on the construction of robust descriptors for the relevant objects. SORT is however largely sensitive to detector quality, and requires several datapoints in order to track with any certainty. The stereo dataset, sadly, did not however have sufficiently long image sequences wherein the same individuals were present to sufficiently test this.

8.4 Extension to multiple view geometry

Naturally, in a practical setting, fixed surveillance cameras or something similar would be much more viable than a single ground-level stereo-based rig. A multiple-view system may also to some degree be more robust to occlusion. This, however, may require engineering robust descriptors from whatever deep-learning based detection model is used in order to establish detection correspondences. One possible approach would for instance to use histogram-based features extracted from segmentation masks constructed from an instance-segmentation based model. Correspondences could then be established between these features through standard distance ratio matching. Alternatively, an array of stereo cameras from multiple views could be used. In this case, a similar approach to what is presented in this report could be used, after which the predictions from each stero-rig could be consolidated using RANSAC such that a robust position estimate would be available for every individual in the scene.

References

[1] Alex Bewley et al. “Simple online and realtime tracking”. In: *2016 IEEE International Conference on Image Processing (ICIP)* (2016). URL: http://dx.doi.org/10.1109/ICIP.2016.7533003.

[2] A. Ess et al. “A Mobile Vision System for Robust Multi-Person Tracking”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR’08)*. IEEE Press, 2008.

[3] *From depth map to point cloud*. https://medium.com/yodayoda/from-depth-map-to-point-cloud-7473721d3f. Accessed: 2021-05-19.

[4] Mathieu Labbé and François Michaud. “RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation: LABBÉ and MICHAUD”. In: *Journal of Field Robotics* 36 (Oct. 2018).

[5] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016.

9 Appendix

Figure 7: Darknet performance prior confidence threshold adjustment

Figure 8: Darknet performance after confidence threshold adjustment

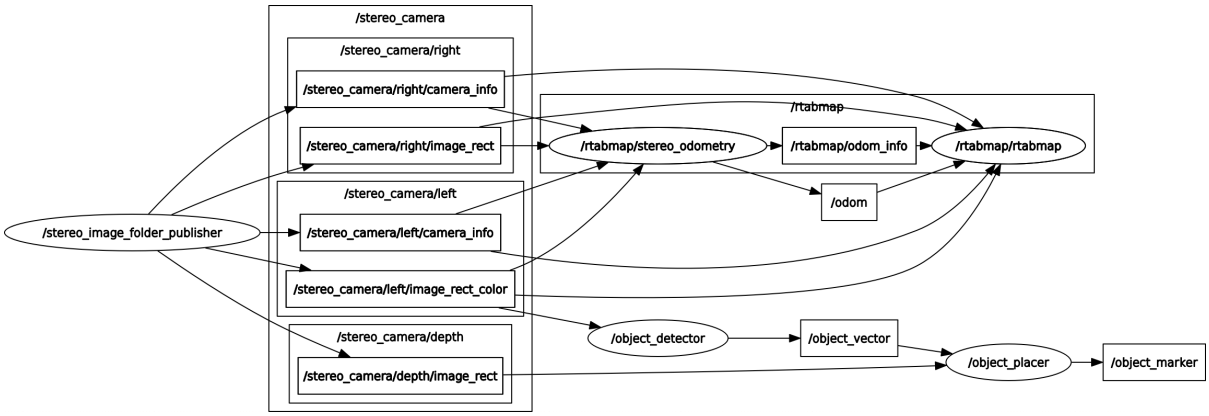

Figure 9: ROS-nodes system overview