Usable surface detection on top-view camera data

using automatic ground truth generation for binary semantic segmentation

MPSYS Design project 2019

Victor Brandt
MPSYS
Chalmers University of Technology
Gothenburg, Sweden
brandtv@student.chalmers.se

Mattias Juhlin
MPSYS
Chalmers University of Technology
Gothenburg, Sweden
juhlinm@student.chalmers.se

Lukas Rauh
MPSYS
Chalmers University of Technology
Gothenburg, Sweden
dlukas@student.chalmers.se

Zhanyu Tuo
MPSYS
Chalmers University of Technology
Gothenburg, Sweden
zhanyu@student.chalmers.se

***Abstract*—A common problem for factory environments is incorporating a flexible assembly-line workflow that does not occupy too much space on the factory floor. For these purposes autonomous vehicles could be used to supply the assembly-lines with materials, but the problem of navigating in the factory then arises. Therefore this project aims to solve the problem of what areas of a factory floor are drivable, using a workflow consisting of background subtraction to generate a ground truth, that is then used to train a neural network. The output could potentially be used as a mapping for which a scheduler or robot control system could work upon to avoid collisions and problems with dynamic obstacles.**

I. Introduction

A. Project Background

One of the upcoming challenges for the vehicle manufacturer AB Volvo is to make the supply for a flexible assembly-line accessible without taking up too much space. This is due to the big number of different parts involved in every different product configuration and they cannot build separate assembly-lines for each of them.

Another incentive for AB Volvo is the fact that factories are continuously working towards having a more flexible production as a whole, which requires a more modular approach to the storage of materials and components used. With that in mind, the old ways of keeping racks of materials close to an assembly line yields a number of problems for a more flexible production, because the time-toll for constantly changing inventory is quite demanding.

B. Literature Study

The main inspiration for the project has been the article by K. Asadi [1], where a very similar approach to the factory surface problem was taken. Binary segmentation was used and a neural network was trained, but the implementation was done locally on the vehicles with a POV-camera as the video source. Despite this, a lot of information regarding network setup and the types of data augmentation techniques could be used in the project. Flipping, cropping and color jitter were used, and the last two are mentioned as amongst the best techniques for improving network accuracy in a paper investigating data augmentation [2].

II. Problem

The purpose of the project is to enable for instance a fleet of autonomous vehicles to differentiate between a drivable surface and a non-drivable surface in a factory setting. This is to be able to use the data for safe navigation and being able to deliver the correct components to a location without endangering workers and/or equipment. Further, the solution to this problem could be used for any type of indoor environment where for instance autonomous robots will be used.

III. Approach

A. Project Approach

The main problem that the project aimed to solve was binary classification of surfaces to deem them drivable or not. This was done mainly through algorithm development using Python, where the workflow consisted of two main steps;

- Data Generation
  - Video Recording
  - Background Subtraction
- Binary Semantic Segmentation

In the project background subtraction was used on pre-recorded data to create a binary ground truth of what is considered foreground and background in a scene, and the data

was then fed to a neural network to perform binary semantic segmentation. Doing this is making way for automatic ground-truth generation which optimize time-consumption. Examples of both methods can be seen in Figure 1 and 2 and background subtraction is explained in [3], semantic segmentation in [4].

The recorded video used for the background subtraction was saved at only 6 frames per second to ease the computational burden on the calculations of the ground truth. This does not impact the performance of the semantic segmentation network because it only sees individual frames without context, and having a lower frame rate does not have any apparent affect. Having a high frame rate mainly yielded many similar frames, thus creating abundant training data.

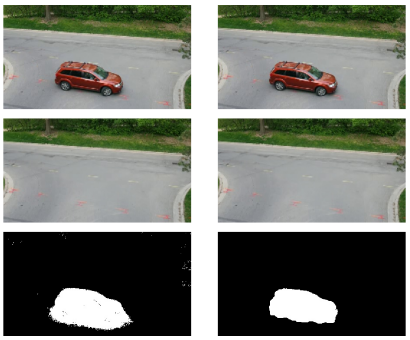

Fig. 1: Background subtraction example [5].

Background subtraction was implemented using an open source real-time computer vision Python library called OpenCV [6]. This library was immensely helpful as it already contains many needed functions and operates with NumPy-arrays, which makes data processing in general much simpler. Several functions was despite this created for the project, both implementing other algorithms and combining existing ones to free up the code and make it easier to debug.

Fig. 2: Multi-class semantic segmentation example [7].

Semantic segmentation can be implemented by feeding the image from the cameras into a CNN encoder consisting of convolutional layers with ReLu activation functions and max-pooling to downscale, in combination with up-sampling decoder architecture using deconvolution similar to what is done in [4]. The output should show if there is ground or obstacle at each pixel.

It can be realized using an open source machine learning Python library like PyTorch [8] or TensorFlow [9]. In the work of J. Long [4], the network GoogleNet[10] was used as the encoder. For the project, the deep nature of a network like GoogleNet does not provide much upside however, due to the simpler task of only binary semantic segmentation. It does only result in a higher computational cost for both training and inference. Therefore a shallower network architecture called U-Net [11] was initially used. U-Net consists of both encoder and decoder. The full approach is summarized in Figure 3.

B. Implementation in Practice

The workflow described above could in practice be used to enhance and optimize performance in a fixed environment/setting. Because of the ease of generating ground truth data through the background subtraction method the whole process can be done in any situation, provided that a background plate can be filmed, and potentially generate better results because the neural network will be trained for exactly the circumstances it will encounter during usage.

It means that as a potential investment, this could be implemented but not put in to active use for a certain time period, where the initial purpose is generating training data. It will lead to a better overall performance of the network output and increase the flexibility of the approach, by learning the certain environment in question.

IV. Data generation

A. Video Recording

The video recording was executed on a number of different locations to ensure that the widest range of floor surfaces as possible were taken into account, i.e. using a diverse dataset to improve the final neural network robustness. For all recordings the camera was mounted onto the ceiling pointing straight down towards the floor in a fixed position, where a number of both fixed and moving obstacles were presented.

Due to the limitations of the project and the fact that the proper equipment was not available, the recordings only contained obstacles such as humans, chairs, backpacks etc. For each scene a background plate was shot without any obstacles at all to make the results of background subtractions as good as possible.

The camera used was an Intel Realsense D435i, which has dual cameras for depth sensing in addition to the regular RGB-camera. The depth data was not used in the project, but could potentially be used in the background subtraction algorithm to improve the accuracy.

B. Background Subtraction

The basis of the project relies on the process of background subtraction on pre-recorded video to create a ground truth in

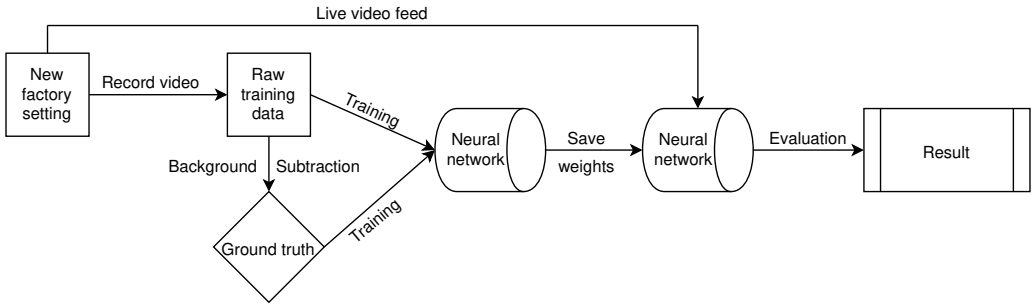

Fig. 3: Flowchart of approach.

order to train the neural network. This pre-processing needs to, with a reasonable accuracy, differentiate between the drivable surface within the current frame and the obstacles that are not a drivable surface.

The whole process is done through four steps.

- Traditional Background Subtraction
- Artifact Removal
- Convex Hull Creation
- GrabCut Execution

Firstly the traditional background subtraction is used, where each individual pixel is evaluated compared to a background plate. This background plate is taken as the median of a recording of just the floor without either moving or static obstacles to ensure that the plate is as good as possible.

The difference in each of the three color channels (RGB) is taken and a manually set threshold is used to decide what difference is needed to regard a pixel as foreground. The reason for the RGB-split is explained in Chapter IV-C. The results from all three channels are then fused with an *or* operation, which means that for a pixel to be regarded as a foreground pixel only one of the three color channels thresholds' needs to be met.

This combined data for each frame becomes a binary representation of all its pixels, where the white pixels are regarded as foreground and the black pixels as background. Each of these binary frames are then used to create convex hulls. Which is done through finding every contour in the binary image, which can be explained as drawing lines around all connected shapes of white pixels, based on the corners of the shapes. Then a convex hull is created with using those contour and the inside is filled with white.

This is done to remove the potential uncertainties and color-mismatches within objects that can cause "holes" or other artifacting due to parts of objects having similar color to the background. As long as the edge of the object has some contrast (which is almost always the case) the detection of the inside of the shape does not matter. The convex hull is used to be more certain of including the whole object even though the base background subtraction might not be perfect. The resulting binary frame is then consisting of convex white shapes that mark the pixels where a foreground object is.

These hulls should be cleaned up because of inevitable noise. This is done through an artifact filter, which consists of a simple algorithm that removes all collections of white pixels that consist of less pixels than the set amount of the function. This is used to remove certain grainy noise and to ease the computational load when implemented before the GrabCut algorithm [12] is used.

These filtered hulls are now used to implement the OpenCV function GrabCut on the original RGB-image. The GrabCut function is initialized at every frame using the mask option, which means that a 2-dimensional matrix (same size as the image) consisting of 0's, 1's, 2's, and 3's is inputed to help annotate what is foreground and background. Here the 0's symbolize all certain background pixels, 1's all the certain foreground pixels, 2's all the probable background pixels and 3's all the probable foreground pixels.

Firstly the 2d matrix was initialized as only containing 2's, and was after that filled with the other values. The hulls are regarded to be probable foreground pixels, because the background subtraction that led to these hulls (after the artifact removal) should make sure to encapsule the essential part of every object. Other than this classification, two RGB-differences like the one explained above was taken with different thresholds.

The first was taken at a higher one than the original, to extract the pixels which are certain to be a foreground object. The second one was instead taken at a lower threshold than the original, and then extracting all zeros in this matrix to extract where the areas that are certain to be background are. It was then implemented in the GrabCut function by setting all numbers in the 2d matrix corresponding to the hulls to 3's, objects in the first RGB-difference to 1's, and the non-objects in the second RGB-difference to 0's.

The result of this process then generates an output that is used as a binary segmentation matrix that is used as the ground truth for training in the next step of the process (see Figure

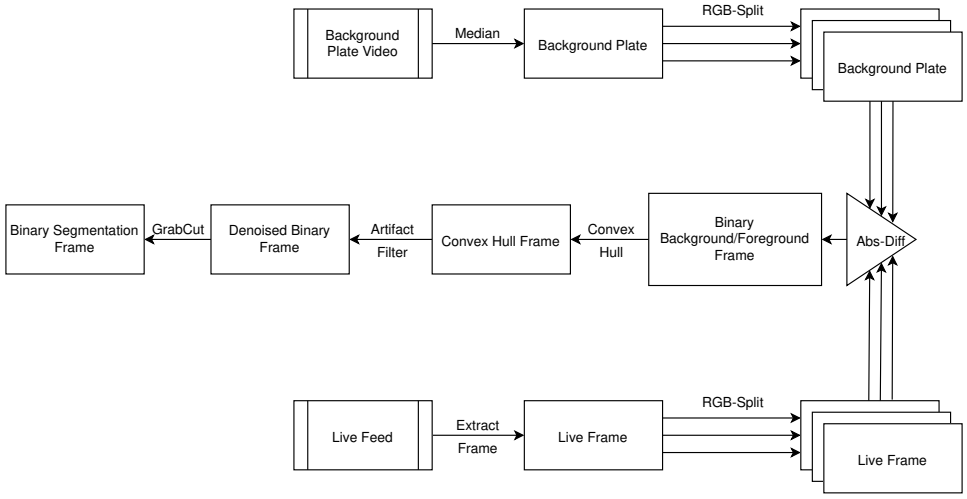

Fig. 4: Flowchart of background subtraction.

Fig. 5: GrabCut applied on footage (top left: input image, top right: raw 3-channel background subtraction, bottom left: convex hulls created, bottom right: result of the GrabCut filter i.e. produced ground truth).

5). An overview of the whole background subtraction process and how the working parts interact can be seen in Figure 4.

C. Old Algorithms and Improvement

During the development of the background subtraction algorithm, several iterations were created and improved by each iteration. This was an essential part of giving the neural network the best possible ground truth to train with.

The first iterations only consisted of the raw background subtraction (as seen in the top right hand corner of Figure 5), and all improvements were centered around tuning the single cutoff threshold for the grayscale difference function to obtain a background/foreground model. This however was an incredibly non-robust model, as lighting changes and similarities of color and patterns of objects and background created quite heavy artifacting. To combat that effect, the RGB image was split into its three color channels, and each channel was treated separately. This in turn yielded a better representation of reality, and greatly improved the performance of differentiating between objects and background that has a similar color value in grayscale.

The problem with this method was that, more often than not, big parts of objects were classified as background, and shadows and lighting changes were detrimental to the performance. Because of that the next iteration was implemented, which introduced the convex hull.

The convex hull removed much of the false negatives that the raw background subtraction introduced, but it was not perfect. For instance (as seen in the bottom left hand corner of Figure 5) if an object is not recognised by the background subtraction accurately enough, the convex hull will consist of multiple non-connected ones which as an end result is not good enough.

That was the reason for the next iteration, which introduced the GrabCut filter. The new version used the convex hull geometry from each frame and conducted the GrabCut algorithm on that area of the RGB image. This was however very computationally heavy, and still suffered from the shifting accuracy of the background subtraction. Therefore the input to the GrabCut filter was changed to be the 4-class 2D matrix that denotes every pixel's likely state.

That change made a huge difference in both computational performance and result, and ended up being the final version. The difference between using the convex hulls and the 4-class matrix for a certain frame is visible in Figure 6, where the middle frame is the convex hull input and the bottom frame the 4-class matrix. It shows that the artifacting is greatly decreased in situations where objects interact and glare is present, which makes the ground truth generation more robust.

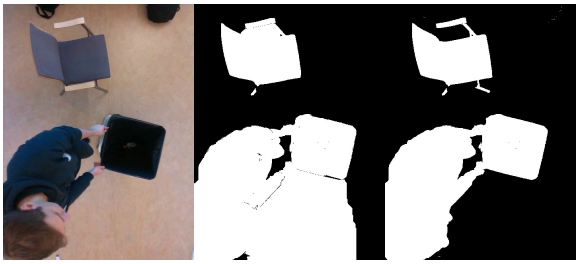

Fig. 6: Difference between the first and second implementation of GrabCut.

V. Neural Network Implementation

The ground-truth data generated by the background subtraction algorithm is used on the recorded video data as input to train neural networks for the task of surface detection, binary segmentation between back- and foreground. The following subsections describe different network architectures used in the project, as well as the training process including key techniques.

A. U-Net

The first network used in the project was the convolutional encoder-decoder architecture of the U-Net, introduced in [11]. Figure 7 shows the architecture of the U-Net network. This network was originally developed for biomedical image segmentation, and can be seen as an development of the initial idea of semantic segmentation proposed by J. Long [4].

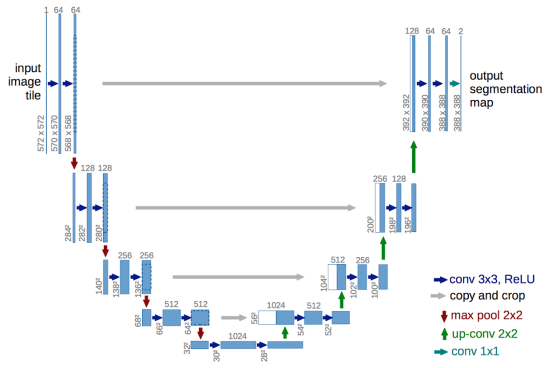

Fig. 7: Architecture of U-Net, from [11].

U-Net is composed of an encoder and a decoder, where the encoder downsamples the input image while the decoder upsamples the output of the encoder. The network contains 9 convolutional layers, 4 max-pooling layers and 4 up-sampling layers.
There are skip connections by concatenating the output of the upsampling layers with the feature maps from the encoder at the same level. So the encoder is classifying major objects differing from the background and the decoder is upscaling the output back to original resolution for enabling proper use of the output image.

B. DeepLab v3+

Another convolutional encoder-decoder tried was a network of the DeepLab v3+ architecture introduced in [13]. It is the latest version of a renowned, powerful state-of-the-art segmentation network, which has often shown good performance in various applications. The network uses atrous separable convolution to reduce the number of parameters significantly. As result the network is able to achieve the performance of a deep network trained on high-resolution input data in a fast and parameter-efficient way. The DeepLab v3+ architecture is shown in Figure 8.Even if the structure and type of DeepLab v3+ and U-net is similar, the network architecture is in detail very different.

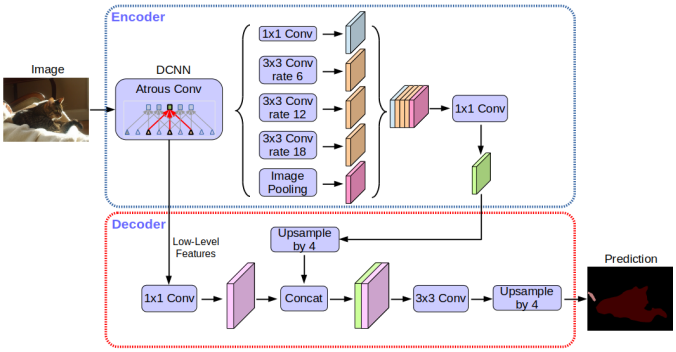

Fig. 8: Architecture of DeepLab v3+, from [13].

C. Training

1) Data Augmentation: Training the neural network on data with different backgrounds, different lightning conditions and foreground objects is essential to achieve useful generalization instead of overfitting to a few of specific backgrounds. To train on more diverse input data without recording a large amount of data the project uses augmentation techniques during the training phase. Horizontal and vertical flip, random cropping and variable color jitter are applied randomly and in random combinations to the input frames. Figure 9 shows a comparison of each of these techniques applied to the same image.

Fig. 9: Data augmentation (top left: original image, top right: flip, bottom left: crop, bottom right: color jitter.

2) Implementation setup: U-Net was implemented in Tensorflow. Besides a second, similar scoring U-Net implementation the DeepLab v3+ network was implemented in PyTorch. To compare both networks a similar training environment was used including an Adam optimizer and cross entropy loss for 10 training epochs.

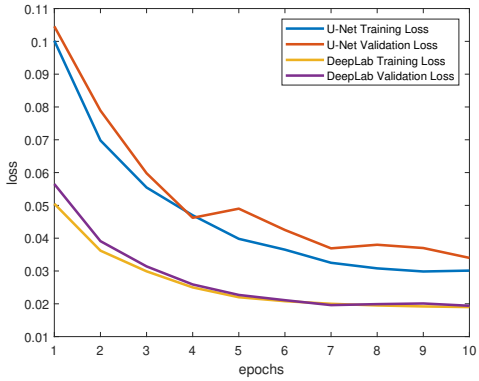

Fig. 10: Training curve of both networks

As it is shown in Figure 10, the loss seems to plateau at about epoch 10 and thus training were stopped.

3) Network performing better than ground truth: Considering that the ground truth is generated by background subtraction, there are sometimes some errors in ground truth, which can be seen in the top right part of Figure 11. However the network sometimes managed to correct the faulty ground truth, and predict a more reasonable output than the ground truth itself, which can be seen in the bottom left part of the same figure. This shows that the neural network can perform good by itself which was the goal of training it.

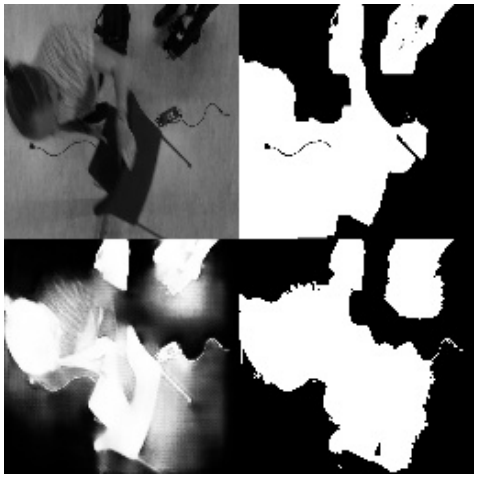

Fig. 11: U-Net process (top left: input image, top right: background-subtraction generated ground truth, bottom left: raw output of U-net, bottom right: binarized output of U-net).

D. Live Testing

To test the performance of the neural networks on more than a recorded test set, a live demo was created to check the real time output of the neural network. For instance, a self-driving mower is driving past the camera in one sequence. A snapshot of the result can be seen in Figure 12, where the red transparent mask shows the binary prediction output on top of the camera input.

Fig. 12: Realtime test showing the binary prediction as red mask over the input.

The mower is not present in our training dataset and it has some reflective surfaces that may disturb the background subtraction but the neural network performance is quite good, performance numbers can be seen in Chapter VI-C. There is also a parameter to adjust the binarizing threshold that could be adjusted according to the condition of lighting to obtain a more accurate result.

VI. Evaluation

A. Comparison of Segmentation and Background Subtraction

Since any solution to the problem stated in Chapter II is worth investigating, using only the background subtraction as output was tested as well. For usage in a live setting, comparing the neural network approach to the background subtraction becomes quite one-sided. To use the background subtraction and get acceptable results, an accurate background plate needs to be used and the thresholds needs to be dynamically tuned.

This still does not ensure good performance, as with lighting changes, restructuring of the observed area or even wanting to move the system to another location, it requires the operator to create another background plate. Another aspect is the computational load, where the background subtraction, due to the usage of GrabCut, struggles to maintain even a 1 FPS frame rate.

In contrast, the neural network approach is the better option for real time implementation in almost every way. The flexibility is greatly improved, as no background plate is needed. It is also trainable for a multitude of scenarios, with the only backside being that the background subtraction method must be used to generate the training data.

By using a neural network, the need of background plates and the sensitivity of changing of the scene is removed. This leads to less variables being able to influence the performance of the algorithm, and thus a more robust approach. The computational strain is also decreased, as an output of > 10 FPS in 640x480 resolution is easy to obtain in real time.

B. Ground Truth

The ground truth produced by the background subtraction algorithm is at times quite accurate but often suffers from artifacts due to certain lightning situations and certain patterns e.g. checkered misleading the algorithm. Using the data as absolute ground truth is not deemed viable. An example of how too much shadows results in a bad output from the background subtraction can be seen in Figure 13, the network output is slightly better but still not optimal.

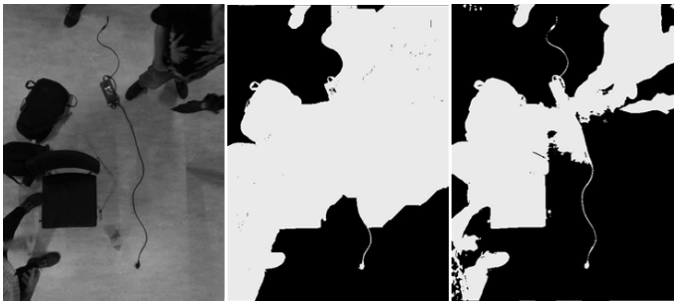

Fig. 13: From left to right, input image filled with many shadows, background subtraction, neural network output

C. Neural Network Performance

Training the U-Net implementation on data from certain rooms and then testing its performance in a new unseen room, yielded a mean Intersection-over-Union (IoU) score over both classes background and foreground of **0.855**. As for using different data augmentation techniques, the mean IoU score of the different networks can be seen in Table I.

TABLE I
Mean IoU-scores for different networks and data augmentation techniques.

	U-Net	DeepLab v3+
No augmentation	0.765	0.850
Random flip	0.753	0.852
Random crop	0.769	0.853
Color jitter	0.843	0.857
All three comb.	0.815	0.854

The differences were greater in the U-Net implementation but a common result was that the biggest improvement of network performance was by using color jitter. As for live performance, the live demo was run using the GrabCut output as ground truth and a comparison of IoU-scores and inference time was made between the different networks and in with basic background subtraction (without convex hulls or GrabCut). The live demo was run in a room used in the training of the network. The results are shown in Table II.

TABLE II
Mean IoU-scores and inference times for different networks and basic background subtraction

	U-Net	DeepLab v3+	BG-sub.
IoU score	0.960	0.971	0.957
Inf. time [s]	0.132	0.096	0.003

The basic background subtraction is really fast, it could run at about 300 FPS while DeepLab runs at around 10 FPS and U-net at around 7 FPS. And even though IoU-score of the basic background subtraction is comparable to the ones of the networks, the method has severe drawbacks as explained in Chapter VI-A. Adding convex hulls and GrabCut would improve the final result but the speed would go down from 300 FPS to a maximum of about 1 FPS.

VII. Conclusion

A. Project Results

As a whole, the final result of the project is very promising. The neural network was quick enough to, when it was ran on a computer with a dedicated graphics card, output > 10 FPS in 640x480 resolution. The output itself was also deemed usable (by visual inspection) despite the lacking variance of the training data, and it even has some robustness against slight shadows and different lighting conditions. With that in mind and with the scope of the project, the end result should be regarded as a success, as it clearly shows off the ease and the benefits of having a self generating ground truth paired with a neural network.

The IoU-scores presented in Chapter VI-C are the best performance measure used in the project but there is a problem when using them. All IoU-scores is calculated in comparison to the generated ground truth, which is not actual ground truth and thus for instance when the network outperform the ground truth as in shown in Chapter V-C3, it will yield a lower IoU-score even though the actual performance by visual inspection is higher.

This explains why for example the IoU-scores for different data augmentation is not varying so much for the DeepLab implementation even though actual performance gain could be seen. Additionally, incorrect ground truth during training can confuse the networks and limit the training performance. It also makes them into values that can't be compared to other work, but in Table II they can be used to verify that the neural networks actually perform on a level comparable to basic background subtraction.

One can also see in the table that the network performs better in a room used for training of the network than an unseen room (**0.855**). There are publicly available datasets for testing binary semantic segmentation network performance but most include either non top-view pictures or medical images which are all very different from our dataset. And thus they were not used for testing.

B. Potential as a Product

This short project has by all means shown the potential of the solution, both by flexibility of implementation, but also from the perspective of centralizing the control of autonomous vehicles under these types of circumstances. The strength lies mostly in the fact that, with enough specific training data, the end result can effectively be used in almost any settings with a high accuracy. The possibility to improve the performance after implementation is also a strength of this approach, as the cost and time it takes to make a general network work well enough is way too high.

Robustness testing is however a important step that needs to be added and performed with care. Due to the closed nature of neural networks one cannot know exactly how the network will react in all circumstances, which creates an unwanted uncertainty. That is something that needs to be resolved before the solution could be turned into a proper commercial product.

C. Further Improvements

The ground truth generation used in the project is quite useful, but could be improved in several ways. A more specific and potentially dynamic tuning could be implemented for thresholds and grouping of convex hulls to avoid classifying shadows as objects while at the same time not loosing track of parts of objects due to their color. Which will in turn lead to a more robust network due to more accurate data for training, and thus a better working end product.

Moreover the variance of training data and obstacle objects used should by all means be increased to ensure a larger coverage and potentially useful recognition of objects in general. However an improvement that has to be adjusted depending on the environment and situations that the product will be exposed to, because good training data in one setting might be detrimental in another. The GrabCut algorithm execution is quite slow and quicker alternatives can be looked at to increase speed of the process.

With that said, the most robust improvement would be to train the network for most environments and objects which, if possible, would remove the requirement on the need of generating unique training data for each new implementation. For instance, to improve performance during lightning changes or environments rich of shadows, training the network in similar situations with proper ground truth might improve robustness. Such ground truth could not be generated by the background subtraction. If this is possible and profitable however was outside the scope of the project.

A potential and quite different approach might also be to generate the ground truth with the GrabCut algorithm, but subsequently let a human examine the frames and correct faulty parts. This leads to a much better accuracy, but requires human interaction and becomes quite tedious. However, it should not be understated that for the time being, this might be the most applicable approach of all, especially for small scale usage where the training data set is not that large.

Further work could also be focused on instance segmentation and estimation of speed and future positions of objects passing the cameras.

REFERENCES

[1] K. Asadi, P. Chen, K. Han, T. Wu, and E. Lobaton, "LNSNet: Lightweight navigable space segmentation for autonomous robots on construction sites," *Data Journal*, vol. 4, no. 1, 2019.

[2] L. Taylor and G. Nitschke, "Improving deep learning using generic data augmentation," *CoRR*, vol. abs/1708.06020, 2017.

[3] A. M. Elgammal, D. Harwood, and L. S. Davis, "Non-parametric model for background subtraction," in *Proceedings of the 6th European Conference on Computer Vision-Part II*, ECCV '00, (London, UK), pp. 751–767, Springer-Verlag, 2000.

[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014.

[5] J. Zhou, "Color separation for background subtraction," 2016.

[6] Opencv.org, 2019.

[7] D. Karunakaran, "Semantic segmentation — Udacity's self-driving car engineer nanodegree," 2019.

[8] Pytorch.org, 2019.

[9] Tensorflow.org, 2019.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[11] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[12] C. Rother, V. Kolmogorov, and A. Blake, "GrabCut: Interactive foreground extraction using iterated graph cuts," *ACM Trans. Graph.*, vol. 23, pp. 309–314, Aug. 2004.

[13] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *CoRR*, vol. abs/1802.02611, 2018.