Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Silas Eichsteller

Multi-image detection and tracking of cracks in ship tanks

Master's thesis in Computer Science
Supervisor: Rudolf Mester
June 2021

NTNU
Kunnskap for en bedre verden

DNV

Silas Eichsteller

Multi-image detection and tracking of cracks in ship tanks

Master's thesis in Computer Science
Supervisor: Rudolf Mester
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Acronyms

CCA Connected Component Analysis.

DoF Degrees of Freedom.

FC Fully Connected.

FCN Fully convolutional Network.

FIFO First In, First Out.

FN False Negative.

FP False Positive.

ITF Interframe Transformation Fidelity.

MC Motion Compensation.

MID Multi-Image detector.

MOT Multiple Object Tracking.

MSE Mean Square Error.

NN Neural Network.

PSNR Peak signal-to-noise ratio.

SID Single-Image detector.

Sammendrag

Denne masteroppgaven omhandler bruk av romlig-temporal informasjon som er tilgjengelig i enkelt bildene til en video, for å forbedre sprekkdeteksjon i skipstanker. Hovedideen er å bevegelseskompensere tidligere bilder til det nyeste bilde i en videostrøm. Dette er for å assosiere rommelig informasjon på tvers av bildene sammen. Dette kan brukes til å etterbehandle detekteringsresultatet fra en semantisk segmenteringssprekkdetektor. Det undersøkes også om et neuralt nettverk som bruker en stabel med multiple bilder kan oppnå bedre resultater enn et nettverk som bare ser på ett bilde av gangen. Dette gjøres både når stabelen er bevegelseskompensert eller ikke. I tillegg er det vist om bevegelseskompensasjon kan brukes til å spore en oppdaget sprekk over etterfølgende bilder i en video.

En modulær arkitektur er beskrevet, som i sin kjerne er basert på en stabel av de n siste mottatte bildene. Bevegelseskompensasjonen tar de forrige bildene i køen og justerer dem så de overlapper mest mulig med det nyeste bilde. Dette er gjort ved hjelp av en 2D-transformasjonsestimering. To etterbehandlingsmetoder er implementert i etterbehandlingsmodulen. I tillegg til den bevegelseskompenserte stabelen, mottar etterbehandlingen en stabel med tilhørende og kompenserte prediksjonskart fra en detektor. I den første tilnærmingen stemmes det over pikslene i de overlappende prediksjonene for å temporal utjevne deteksjonsresultatet. I den andre tilnærmingen blir disse prediksjonskartene satt inn i et neuralt nettverk (NN). Totalt fire NN med forskjellige egenskaper er utviklet og testet. I den multiple bilde detektor modulen, blir hele bilde stabelen satt i ett neuralt nettverk. Denne tilnærmingen er prøvd ut ved hjelp av den utviklede bevegelses kompenseringen og uten å for å se om et slikt nettverk kan lære seg å bevegelses-kompensere på egenhånd. Sporingsmodulen grupperer sammen og instansierer oppdagede sprekkpiksler før de spores over påfølgendebilder, ved hjelp av piksel-til-piksel-tilknytning gitt av bevegelseskompensasjonen.

Både etterbehandlingen og multibildedetektoren viser temporal mer stabile resultater enn en enkelt bildedetektor. Resultatene antyder også at en multibildedetektor som ikke bruker bevegelse kompensering kan lære seg å bevegelses-kompensere på egenhånd. Den implementerte sporing gir også tilfredsstillende resultater. Imidlertid er de fleste modulene veldig avhengige av påliteligheten til bevegelses kompenseringen. Mens bevegelses kompenseringen generelt ser ut til å fungere bra, forekommer det noen ganger noen unøyaktigheter. Disse unøyaktighetene kan få store konsekvenser for resultatene av modulene som bruker bevegelses-kompenseringen.

Abstract

This master thesis is about using spatial-temporal information available across the frames during a video sequence to improve crack detection in ship tanks. The main idea is to motion compensate previous frames to the most recent frame in a video stream to associate the spatial information across frames. This can be used to post-process the detection result from a semantic segmentation crack detector. It is also explored whether a Deep Neural Network with a Multi-Image stack as input can outperform a Single-Image detector. This is tested both with motion compensating the stack, and without. Additionally, it is shown that motion compensation can be used to track a detected crack over subsequent frames.

A modular pipeline is described, which at its core is based around a motion compensated stack of the n most recently received frames. The motion compensation takes the previous frames aligns them with the newest frame, using 2D transformation estimation. Two post-processing approaches are implemented in this project. In addition to the motion-compensated stack of frames, the post-processing receives a stack of the associated pixel-wise segmentation output (detection map) from a detector (which also is compensated). The first approach is a simple handcrafted voting approach, which uses the alignment of pixels in the detection maps to temporally smooth out the detection result. In the second approach, these detection maps are put into a Neural Network (NN). A total of four NNs with different properties are developed and tested. In the Multi-Image Detector (MID) module, the entire stack of frames is put into a NN. This approach is evaluated with and without using motion compensation MC to see whether the MID can learn to motion compensate on its own. The tracking module is the last module in the pipeline. It groups together and instantiate detected crack pixels before tracking them over subsequent frames, using the pixel-to-pixel association provided by the motion compensation.

Both the post-processing and the Multi-image-detector show more stable results than a Single Image Detector on its own provides. The results also suggest that the MID that does not use the motion compensation learns to motion compensate on its own. The implemented tracking also provides satisfying results. However, most of the modules are deeply dependent on the reliability of motion compensation. While the motion compensation overall seems to operate smoothly, some inaccuracies sometimes occur. These inaccuracies can have massive consequences for the results of the modules using the motion compensation.

Preface

This Master Thesis has been conducted at the Department of Computer Science at the Norwegian University of Science and Technology (NTNU), in collaboration with Det Norske Veritas (DNV). The primary research and development was done during the spring of 2021. The master thesis is a continuation of a specialization project which was conducted during the fall of 2020. It has been a fantastic experience to do such a large-scale project, especially with all the great support I got from the industry and the academic side.

First of all, I want to express my appreciation for the people at DNV, giving me both insides into their research and access to data and hardware. Geir Hamre has been my main point of contact at DNV and has helped me set up the developer environment and get the data I need. I also want to thank André Ødegårdstuen, which together with Geir, has participated in bi-weekly meetings, where we discussed the current progress and findings of my research. Both of them have shown interest in my research and have, throughout this project, given me constructive feedback. I want to thank Jing Xie as well, which gave me a lot of insights into her work and feedback during my specialization project.

On the academic side, I want to give special thanks to my supervisor Rudolf Mester, which has supported me greatly during this thesis. He has given me a lot of insights into classical computer vision approaches and overall great ideas on how to solve occurring problems. I am impressed over how consistently Rudolf answers emails, both on regular weekdays and holidays. He has often taken the initiative to talk to me about my thesis via Teams, even when it is a Saturday night.

Finally, I would like to thank friends and family for their support during the long days I worked on this Master thesis, especially when it comes to pushing me to sometimes disconnect from my studies and go outside.

Contents

1. Introduction **10**
- 1.1. Background and Motivation . . . 10
- 1.2. Goals and Research Questions . . . 11
- 1.3. Contributions . . . 11
- 1.4. Thesis Structure . . . 12

2. Background Theory **13**
- 2.1. Fundamental of object detection by Deep Neural Networks . . . 13
- 2.2. About different Deep Learning techniques . . . 14
  - 2.2.1. DeepLabv3 . . . 15
- 2.3. Fundamentals of motion compensation . . . 17
  - 2.3.1. 2D Transformation . . . 17
  - 2.3.2. Feature extraction in frames . . . 20
  - 2.3.3. Descriptor based keypoint matching . . . 20
  - 2.3.4. Estimating transformation parameters from associated keypoints . . . 22

3. Method **25**
- 3.1. A brief overview of the pipeline . . . 25
- 3.2. Motion Compensation . . . 26
  - 3.2.1. Equation for updating transformation matrices in a stack . . . 28
- 3.3. The post-processing module . . . 30
  - 3.3.1. Post-processing with voting . . . 31
  - 3.3.2. Post-processing with Neural networks . . . 32
  - 3.3.3. Training data and augmentation . . . 33
- 3.4. The Multi-Image Detector module . . . 37
  - 3.4.1. Training data and augmentation . . . 37
  - 3.4.2. MID module implementation . . . 40
- 3.5. The Tracking module . . . 41

4. Experiments and Results **43**
- 4.1. Experimental plan . . . 43
  - 4.1.1. Available real videos . . . 43
  - 4.1.2. Plan for testing the motion compensation . . . 44
  - 4.1.3. Plan for testing the post-processing module . . . 45
  - 4.1.4. Plan for testing the MID module . . . 46
  - 4.1.5. Evaluation metrics for the post-processing module and the MID module . . . 46
  - 4.1.6. Plan for testing the tracking . . . 47
- 4.2. Results . . . 48
  - 4.2.1. Results for motion compensation . . . 48
  - 4.2.2. Results for different voting parameters . . . 51
  - 4.2.3. Results for post-processing module and MID module . . . 52
- 4.3. Interesting observations in the visual evaluation . . . 54
  - 4.3.1. Tracking results . . . 56

5. Discussion **57**
5.1. Motion Compensation Discussion . . . 57
5.1.1. Best 2D transformation type . . . 57
5.1.2. Difference between naive and update approach . . . 58
5.1.3. Self evaluation of the motion compensation . . . 59
5.2. Evaluation of the post-processing module . . . 60
5.2.1. Post-processing module, voting behavior . . . 61
5.2.2. Behaviour of the Post-processing Neural Networks . . . 62
5.2.3. Evaluation of the MID module . . . 65
5.2.4. Tracking . . . 66

6. Conclusion **68**

7. Proposals for Future Work **70**
7.1. Improving motion compensation . . . 70
7.1.1. Motion compensation performance measurement . . . 70
7.1.2. Motion compensation with Neural Networks . . . 70
7.2. Improving the pipeline . . . 70
7.2.1. The ideal stack depth . . . 70
7.2.2. Taking frames that are forward in time into consideration . . . 71
7.2.3. Using output from previous post-processing as input . . . 71
7.3. Improving the tracker . . . 71
7.4. Improving the quality of training data . . . 72
7.4.1. Semi-automatic hand labeling . . . 72
7.4.2. Using Box labels . . . 74
7.4.3. More augmentations for Single-Image detectors . . . 74

Bibliography **75**

Appendix A. Developed models **78**
A.1. Small Network . . . 78
A.1.1. Small model architecture . . . 78
A.2. Small Network with Skip connections . . . 79
A.2.1. Small model with skip connections architecture . . . 80
A.3. UNet based NN for post-processing . . . 80
A.4. DeepLabv3 . . . 80
A.4.1. Post-processing . . . 81
A.4.2. SID . . . 81
A.4.3. MID . . . 81

Appendix B. Further results and discussion **82**
B.1. About the attached videos . . . 82
B.2. Other interesting observations . . . 83
B.3. Further results for different 2D transformation types . . . 86
B.3.1. Further results for execution time of all the approaches . . . 87

Appendix C. Numbers used for metric calculation **88**

Appendix D. Singular value decomposition (SVD) **89**
D.0.1. SVD example . . . 89

Appendix E. More about ORB **91**

Appendix F. Video presentation **95**

1. Introduction

1.1. Background and Motivation

This project is done in collaboration with DNV, an international company that focuses on quality assurance and risk management. DNV is a world-leading classification society and a recognized advisor for the maritime industry. One of the tasks DNV faces is the inspection of ship tanks. Defects such as cracks and corrosion of the material inside the tanks can lead to risks when remaining undetected. For a long time, the inspection task has been conducted manually, which is both expensive and hazardous. In recent years DNV has started to research the use of drones and artificial intelligence to aid in this task. By utilizing this technology, inspections can potentially be conducted faster, cheaper, and more accurately. Integrating drones in the inspection workflow will also lead to a safer working environment.

For the detection of cracks in ship tanks from drone camera video footage, DNV has developed different crack detectors. The detection results from a couple of videos, by a particular detector, were made available for this project. The detector classifies each pixel in a frame to either belong to a crack or not a crack. It only looks at the most recent frame in a video stream during inference. However, all the information in previous frames in the stream are not utilized. This project will investigate the potential of using previous frames in a video stream to make the detection in the current frame more robust.

As the drone flies, the camera moves. The objects in the scene are stationary, and are not moving by themselves. Instead, the entire view is moving as the drone moves. In order to utilize the information in a previous frame, an association between its spatial information and the current frames' spatial information, needs to be established. Suppose the view would be stationary. i.e., the camera would not move; each pixel location in the previous frame could then be mapped to the exact pixel location in the current frame. To be able to do such a direct mapping, despite the camera movement, the previous frame needs to be *motion compensated* so that it matches the current frame. This type of motion compensation is done to achieve a pixel-to-pixel association between subsequent frames is fundamental for the different modules developed in this project.

The first module introduces a post-processing step that utilizes the detections done in the previous frames to improve the detection in the current frame. It is assumed that a detection result is represented as a label map, where each pixel has a class label. The same compensation used to match a previous frame to the current frame can be applied to its label map. The pixels in the label map from the previous detection can then be associated with pixels from the current detection. A stack of such label maps originating from detections done in the previous frames makes this already extracted information available to the current detection. There are several ways all the information in the stack can be combined. This project covers both a pixel voting approach and the use of Neural Networks. The ultimate goal is to smooth out inconsistent detection results, making the detections temporally consistent and less "flickery" in-between frames.

The detection model DNV currently uses is a *Single-Image Detector* (SID), meaning it only extracts information from the current frame. The idea for the second module is to expand the input layers of such a model to take in multiple images. Such a *Multi-Image Detector* (MID) is capable of looking

back at multiple previous frames when making a prediction. While the first module is limited by the base detector's capability to extract information, this module tries to improve the extracting process. Even between two consecutive frames, the information available can differ significantly due to, i.e., spectral reflections in the 3D structure of the scene, changing lights and shadows, motion blur, and video compression artifacts. For example, if there is a lot of motion blur in the current frame, the information from the previous frames can still be used when making a prediction. Variation in appearances in frames could make the MID even more robust. While the SID is an image-segmentation model used on video input, the MID aims to be a video-segmentation model. In the first step, the MID uses a stack of motion-compensated images as input. This enables the model to mainly focus on extracting information without the need to worry about the association between frames. However, the idea of letting the network itself handle the motion compensation is also explored.

Motion compensation is mainly used to solve the association problem between the pixels in the current and the previous frames. Note that this association assumes that the only expected movement in the view originates from the camera's movement. Basically, all the pixels in-between frames are tracked as long as they are in the view. A detected object is a subset of all the pixels in the current frame. Therefore the object itself can also be tracked in-between frames. This is what the third and last module aims to achieve.

1.2. Goals and Research Questions

Goal *Use subsequent frames in a video sequence to improve the detection result of an arbitrary detector, make an improved detector with multi-image input, and track detections*

Research question 1 *Can motion compensation be used for post-processing to smooth out detection results from an arbitrary detector which uses a video sequence as input*

- Can this give more temporally stable detection results?
- Is handcrafted post-processing based on pixel voting sufficient or does a Neural Network specialized in the task have a better performances
- What architecture should such a NN have, and how deep does it need to be?

Research question 2 *Is a detector that uses a motion-compensated stack of previous frames over using a single input frame more reliable?*

- Can such a Multi-Image detector learn to motion compensate on its own?

Research question 3 *Can motion compensation be used for tracking a detection over multiple frames in a video sequence?*

- How reliably can a detection be tracked?
- Are there many identity switches?
- What to do if detection is lost for multiple frames?

1.3. Contributions

This project presents a pipeline which uses motion compensation to improve detection results and track cracks in ship tanks, given a video stream input. This is done by combining classical computer

vision approaches with Neural Networks. A post-processing module, a Multi-Image Detector (MID) module, and a tracking module are developed.

A motion compensation procedure based on 2D transformation matrix estimation is introduced. This creates a stack of previous subsequent frames, which includes the respective frame's associated detections and crack instances. It is mainly this stack the developed modules rely on.

The post-processing module can be used to retrofit already existing detectors to become more temporally stable. This module introduces a handcrafted approach as well as a Neural Network (NN) based alternative. The handcrafted approach can be implemented fast. However, the Neural Network can potentially learn to compensate for the specific inaccuracies the underlying base detector has. Four Neural Network architectures are developed/modified, to work with the post-processing step. These are; a small NN, a small NN which introduces skip connections, a UNet based NN, and a DeepLabv3 based NN.

Limited real-life crack videos with labeled ground truth to train the NNs with is available. Therefore, a perlin noise based data augmentation strategy is introduced. The augmentation emulates a motion compensated stack of subsequent detections using single ground truth label maps. The augmentation generates False Positive and False Negative detections in order to emulate a theoretical crack detectors behaviour when it is used on a video-sequence.

The MID module stands as an alternative approach to already existing Single-Image detectors (SID). A Multi-Image detector takes inn multiple images, to take advantage of the appearance variation of pixels in subsequent frames. Both a MID model that uses the motion compensation and a MID model that learns to motion compensate on its own, are trained.

To make up for the lack of labeled video sequences available, a pipeline to generate synthetic videos from labeled still images is developed. Different augmentation strategies are also presented. These are to give MIDs more variety in the training data and emulate the behavior of real-world video sequences.

The Tracker module uses the pixel-to-pixel association provided by the motion compensation in order to track detected cracks over subsequent frames.

1.4. Thesis Structure

In chapter 2 the fundamentals of object detection and motion compensation is presented. Chapter 3 goes through the implementation of a Motion compensation and the three developed modules that are based on it. The goal of each of the three modules is to answer each of their respective research questions. Chapter 4 presents how the experiments are conducted and the results. These results are discussed further in chapter 5 and a conclusion is drawn in chapter 6. Some thoughts on how to improve the results in an eventual continuation of this project are given in chapter 7.

2. Background Theory

This chapter contains some background information about object detection and motion compensation.[1]

2.1. Fundamental of object detection by Deep Neural Networks

Over millions of years of evolution, human eyes and brains have evolved the ability to capture and interpret light rays. The field of computer vision tries to automate tasks and gain a similarly high-level understanding, like the human visual system can do, using digital sensors. This section will mainly focus on the field of object detection, as it is the most relevant for this thesis. Object detection is a core discipline in the field of computer vision. The problem is about classifying and locating certain objects in images. The location of the identified objects can be interpreted in different ways. This includes drawing bounding boxes around the objects or label every pixel in an image with its associated class. The task of object detection can be divided into four subcategories, sorted here after increasing a high-level understanding of the received images. These subcategories are depicted in Figure 2.1. To easier differentiate between the categories, the objects themselves are divided into supercategories; namely *things* and *stuff* [2]. The first one includes objects with well-defined shapes, i.e., a car, a person. Stuff, on the other hand, usually includes connected background surfaces such as grass and sky.

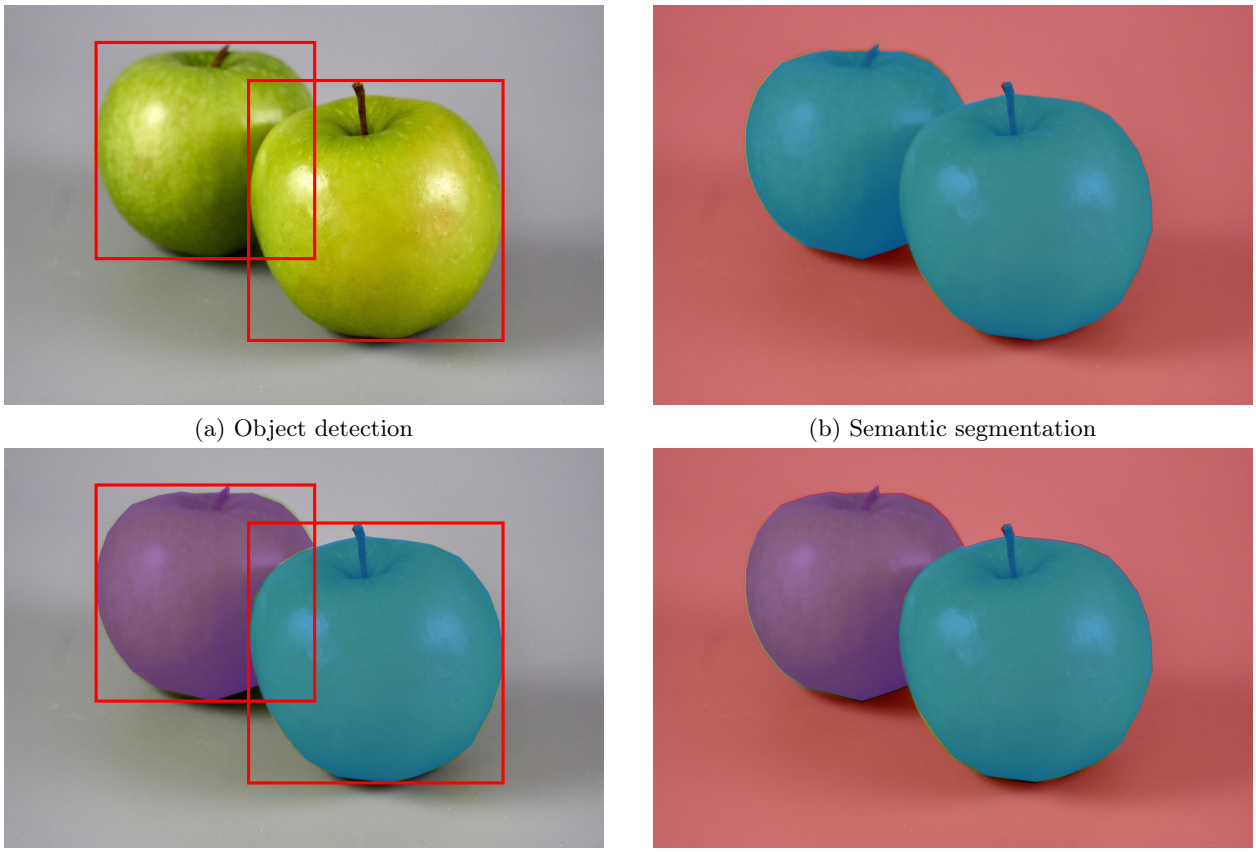
The first subcategory is *object localization/classification*. In this task, explicit things are recognized, classified and the position of these things is marked with bounding boxes. Some of the most commonly known algorithms that do this task are Single Shot Multibox Detector [16] and Unified, Real-Time Object Detection [22].

In *semantic segmentation*, the stuff category is central. A pixel-wise classification is done on every pixel, labeling each pixel with a semantic class. This includes both things and stuff. In Figure 2.1b all the pixels that belong to an apple are colored blue, while all the background pixels are colored red. If a class is not known for a semantic segmentation model, it falls into the background category, which is a subcategory in stuff. The colors indicate the different class labels.

Instance segmentation is about gaining an even higher level of understanding of objects in an image. Each object that belongs to the same class is treated as a separate entity. Every pixel belonging to a thing is associated with both a class value and an instance of that class. Figure 2.1c shows two class instances of apples, where the different colors of the two apples indicate a different class instance. The red box is also there to indicate the different class instances.

In *panoptic segmentation*, the image is no longer divided into the "stuff" and "things" categories but is instead tried to be understood as a whole, including both things and stuff. It is, in many ways, a combination of instance and semantic segmentation (see Figure 2.1d). A pixel-wise classification is done on every pixel. If a pixel belongs to a thing, it additionally is instantiated.

[1] Sections 2.3.1, 2.5, 2.3.3, and 2.3.4 are taken from the Specialisation Project Report (Eichsteller 2020)



(a) Object detection

(b) Semantic segmentation

(c) Instance segmentation

(d) Panoptic segmentation

Figure 2.1.: Examples of different subcategories of object detection.

2.2. About different Deep Learning techniques

In the field of object detection, Neural Networks have achieved remarkable results over the past years, which has led them to replace many of the classical computer vision approaches. The primary algorithm that has spearheaded this shift is the Convolutional Neural Network (CNN). When an image is fed through a CNN, relevant filters are applied to capture spatial dependencies in the image. In earlier object detection approaches, such filters were engineered by hand. In a CNN, however, these filters are learned.

Classical CNN architectures, such as AlexNet [14] and VGG [29], have convolutional and pooling layers, starting from the input layer, and a Fully Connected (FC) classification layer at the end. This works great as long as the input images are all of the same sizes. Images that do not fit the fixed size need to be resized. Resizing the images can leave much distortion in the features of the image, especially if the aspect ratio differs as well. These types of architecture are mostly suited for classification tasks and not for pixel-wise semantic segmentation.

Fully convolutional Network (FCN) [17] marked a milestone in the field of semantic segmentation. By switching out the FC layer of the classical CNN with a Deconvolution layer (aka. transposed convolutional layer), the extracted hierarchic features can also be used for semantic segmentation. These layers perform an upsampling, with a given stride and padding used to scale up the feature maps. This enables the network to classify each pixel while it is trained fully end-to-end. Because no FCs are used, images with different resolutions and ratios can be run through the network. By avoiding the use of dense layers, FCNs have fewer parameters, making them faster to train.

One of the main downsides of the FCNs is that fine-grained spatial information can be lost in the down-sampling path. Convolutions are only connected to local regions from the input and therefore lack a global context. FCNs can be significantly improved when enabling them to access global context knowledge. One way this can be done is by modifying the network architecture. In UNet [23] a lateral skip connection between the Encoder (down-sampling path) and the decoder (up-sampling path) are added, by concatenating the feature maps. This gives the last layer information from the first feature map, giving it more context to process. Another more recent development is the use of Atrous Convolutional Layers (aka. Dilated Convolutions). Atrous Convolutions allow the convolutional layers to get input from an exponentially bigger field of view without any added computational overhead. In DeepLabv3 [4], Atrous Convolutions are used with different dilation rates in a Pyramid Pooling module. This is discussed in greater detail in the next subsection.

2.2.1. DeepLabv3

DeepLabv3 is the main semantic segmentation architecture used in this project. Like most other semantic segmentation networks, this architecture utilizes an FCN Encoder-decoder architecture. In DeepLabv3, features are extracted from a backbone network (i.e., ResNet, VGG, DenseNet). The backbone is a part of the encoder, which extracts features from the input image. However, the spatial resolution of the feature maps decreases the deeper down in the down-sampling path, the information goes. In a standard encoder-decoder network, the feature maps are scaled up to the original resolution of the input image in the up-sampling path. This is either done with deconvolutional layers or interpolation. When upscaling the low-res feature maps from the output of the encoder, fine details from objects can get lost, and borders can get blurred. As mentioned in the previous section, a way to combat this loss of information is to establish lateral connections between the encoder and decoder.

An alternative is NOT to increase the output stride, i.e., making the feature maps bigger so that they contain more spatial/location information. However, this is not feasible in deep networks, as the increased number of parameters makes it too computationally expensive. With Atrous convolutions, the stride can be kept constant, with a larger receptive field. This can be done without increasing the amounts of parameters, thereby keeping the computational cost low. The resulting larger feature maps are great for semantic segmentation as objects of varying scale can be better preserved deeper down in the network.

Atrous convolutions have an extra parameter r, which is the atrous rate. The atrous rate is the stride the input signal of the feature map is sampled with. Basically, $r-1$ dictates how many zeros are inserted between two consecutive filter values along in each spatial dimension. In the case of $r = 2$, 1 zero is inserted between each adjacent filter value in all spatial dimensions. The idea behind using atrous convolutions is that the field-of-view can be modified by only changing r instead of learning additional parameters. This gives flexibility to how dense computed features are. Consider a two-dimensional input feature map x, like the blue-colored region in Figure 2.2, where atrous convolution is applied using a filter w and the atrous rate r. The value for each location i on the outputted feature map y is given by:

$$y[i] = \sum_{k} x[i + r\dot{k}]w[k]$$

To summarize the DeepLabv3 architecture, firstly, features are extracted from the backbone network. Atrous convolutions are used in the last couple of blocks of the backbone to control the size of the feature map. Atrous Spatial Pyramid Pooling (ASPP) is placed on top of the backbone to obtain multi-scale context information. ASPP samples the output of the backbone in four parallel atrous

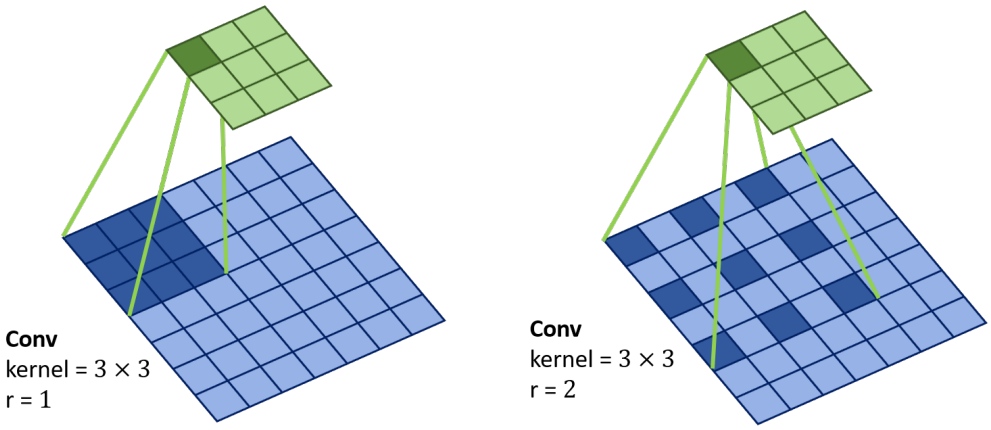


Figure 2.2.: Atrous convolutions with different rates. A $r = 1$ on a Aterous convolution corresponds to a standard convolution. By using a larger rate, the models field-of-view becomes bigger, enabling it to extract information from a variety of scales.

convolutions with different rates. This enables the network to classify different objects at different scales. The ASPP was first introduced in DeepLabv2. In DeepLabv3, the ASPP changed to include both batch normalization and the ability to extract image-level features. The latter is done by applying a global average pooling on the last feature map of the backbone, extracting global context information. The results are concatenated along the channel, and a 1×1 convolution is used to get the final output. An overview of the DeepLabv3 model architecture is depicted in Figure 2.3.

In this thesis, DeepLabv3 is modified to be used in different tasks. This is described more in detail in section 1. DeepLabv3 comes shipped with PyTorch, making it convenient to implement in this project. While DeepLabv3 is no longer the most state-of-the-art in image segmentation, it is sufficient for the purpose of this thesis.

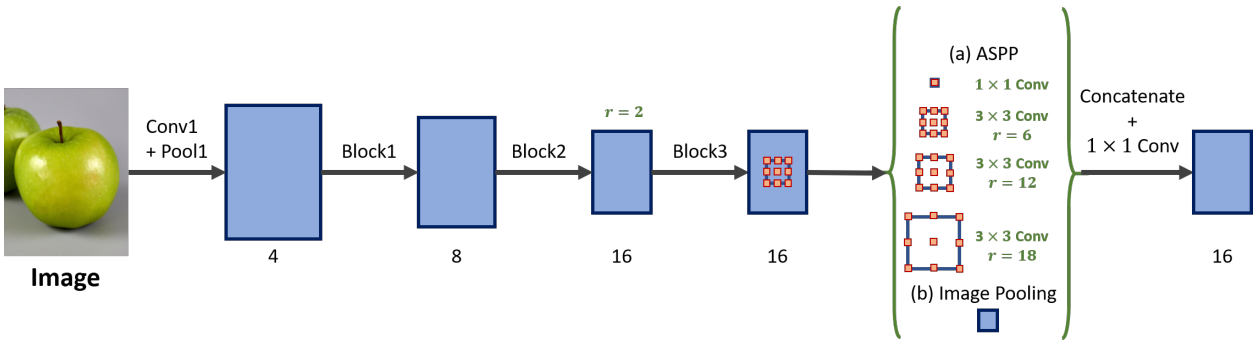


Figure 2.3.: DeepLabv3 model architecture

2.3. Fundamentals of motion compensation

The task of motion compensation is often found in video compression. In a video, the difference between frames is often the result of either camera movement or objects moving. A lot of the information in the current frame can be described as a transformation of the previous frame. In fact, this can be done with frames even further back in time. However, the more movement there is, the less accurate this becomes. Finding and applying those transformations for either past or future frames is known as Motion Compensation (MC).

Since it is assumed that no gyroscopic data from video cameras are available, the compensation needs to be calculated. In video compression, motion compensation is often done by dividing the frames into multiple blocks. A transformation between the blocks in the frames is then estimated. However, this project uses a more classical computer vision approach, namely calculating the 2D transformation between entire frames. Other techniques such as the ones used in compression, 3D transformation, or neural networks are beyond the scope of this project. The 2D transformation approach should be sufficient to answer the research questions while keeping the project's scope manageable. The goal of the motion compensation is to align previous frames to the current frame. In many ways, this is a form of video stabilization, intending to eliminate the camera's movement.

2.3.1. 2D Transformation

Consider an image to be represented as a 2D array. Each element in the array contains a pixel value, i.e., a color RGB value or a pixel intensity value (for grayscale images). Since this is a 2D array, each pixel has a coordinate (x, y) that can be looked up. When a transformation is applied to an image, all pixel positions are changed, allowing the image to change, i.e., its translation, rotation, scale, perspective, or a combination of all of the above. This process can be done by taking the dot product between an input image and a 3×3 transformation matrix. In motion compensation, frame A needs to be transformed to match frame B. To find a transformation matrix that does this well becomes therefore essential. Several kinds of transformation matrices can be used, each with a different amount of parameters and Degrees of Freedom (DoF). Note that the transformation matrices described here are only expected to perform well when operating on planar objects.

Homogeneous coordinates

It is mathematically impossible to take the dot product between a 2D-point and a 3×3 transformation matrix. In order for the transformation to work, each 2D-point needs to be described as a 3D vector:

$$\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Here $w = 1$ is the 3rd coordinate in the 3D-vector. The 3D-vector can also be converted back to a 2D-point; $(x, y, w) \rightarrow (x/w, y/w)$. From this it becomes evident, that a 2D-point can be expressed as different 3D-vectors. E.g. the 2D-point $(3, 1)$ can be represented with 3D-vectors such as $(3, 1, 1)$, $(6, 2, 2)$, $(9, 3, 3)$. It can be observed that if $w = 0$ the 2D-point would go towards infinity. This also means that the 3D-vector $(0, 0, 0)$ can not exist [19].

Affine transformation

Affine transformation geometrically distorts an image while still preserving parallel lines. However, distances, and angels, on the other hand, will not be preserved. There are several kinds of transformations included in the affine transformation category.

In *translation*, all the points are displaced the same direction and distance. The translation calculation for a single point in 2D-space is:

$$\mathbf{T}_{translation}\vec{p}_1 = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} = \vec{p}_2$$

The translation matrix is based on the identity matrix but with the translation values t_x and t_y specified on the top two rows, on the most right column. Here the point that becomes translated is represented as a homogeneous vector $\vec{p}_1$.

The *rotation matrix*, describes a circular transformation, where an image gets rotated around an axis or point. The new coordinates for a single point in 2D-space after rotating θ degree around the origin is given by:

$$\mathbf{T}_{rotation}\vec{p}_1 = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} xcos(\theta) - ysin(\theta) \\ ycos(\theta) + xsin(\theta) \\ 1 \end{bmatrix} = \vec{p}_2$$

The *scaling transformation* is a linear transformation that gives an image a zoom in/out effect. The scaling factors s_x and s_y determine how much an image scales in each respective axis. If $s_x = s_y$ the scaling is considered uniform.

$$\mathbf{T}_{scaling}\vec{p}_1 = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} xs_x \\ ys_y \\ 1 \end{bmatrix} = \vec{p}_2$$

In *shear transformation*, all the points that are on a given line L remain stationary. The remaining points are shifted proportionally to their perpendicular distance to L. The area of the image remains the same. The shearing factors λ_v and λ_h determine the amount of shearing on the vertical and horizontal axis, respectively.

$$\mathbf{T}_{scaling}\vec{p}_1 = \begin{bmatrix} 1 & \lambda_v & 0 \\ \lambda_h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + y\lambda_v \\ y + x\lambda_h \\ 1 \end{bmatrix} = \vec{p}_2$$

All the described transformation types can be combined into a single matrix. A subcategory of affine transformation is *similarity transformation*, which includes a combination of translation, rotation, and uniform scaling. This transformation has 4 degrees of freedom in a 2D plane. The super category affine transformation includes shear and aspect ratio and has 6 DoF. Because of the translation part, the origin of an image after transformation does not necessarily map to its original origin. Figure 2.4 shows an overview of the different transformation types that are under affine transformation. The dark blue shape represents an image before transformation, and the red shape is the image after transformation. Note that a 2D affine transformation matrix can also be described as a 3×2 matrix (the top 2 rows) when calculating a non-homogeneous 2D-point.

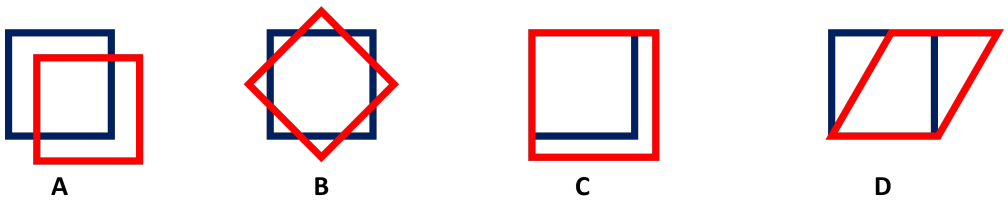


Figure 2.4.: Example of the different transformation types under affine transformation. **A.** translation, **B.** rotation, **C.** scale, **D.** shear

Homography transformation

Consider two images containing the same plane object but from different perspectives. There are four corresponding points marked on the two images in Figure 2.5. Each colored dot in image 1 corresponds to the dot of the same color in image 2. The mapping between those points is done with a homography transformation.

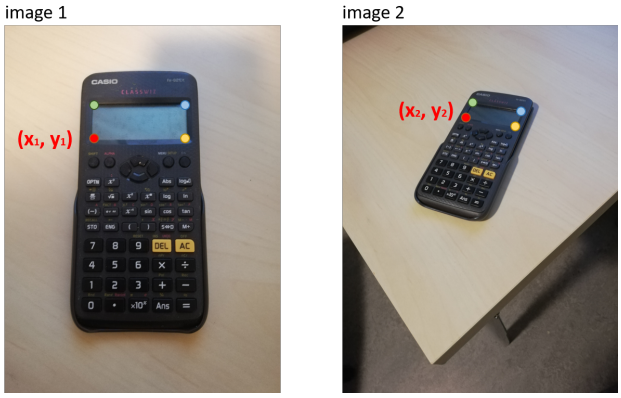


Figure 2.5.: Two images with different perspective

The homography transformation matrix is a 3×3 matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

Considering the homogeneous corresponding red points in the two images (x_1, y_1) and (x_2, y_2), the homography matrix $\mathbf{H}$ maps them like this:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

If $\mathbf{H}$ is known, this calculation can be done for all the points in an image. The homography holds up as long as the corresponding points used to calculate the parameters are in the same plane in the real world. As there are eight unknown variables in the homography transformation matrix, it has 8 DoF [20].

2.3.2. Feature extraction in frames

To find the parameters needed to transform the view from one image to another, a set of corresponding points between them need to be found. Keypoints are points of interest in an image like corners and edges. Example of algorithms keypoint extraction algorithms are Harris Corner Detector [12], the improved Shi-Tomasi Corner Detector [13] and Features from Accelerated Segment Test (FAST) [25]. All of these can also be found in the OpenCV library. In this project, ORB is used for feature extraction, which is based on the FAST algorithm.

FAST overview

A pixel p is selected in an image to identify if it is a good point of interest or not. A small circle with a circumference of k-pixels and with p in its center is considered. The brightness of those k surrounding pixels is then compared and sorted into three classes: lighter, darker, or similar to p. When more than half of the pixels are either in the darker or brighter class, p is selected as a key point. FAST does not have any corner detection, but it naturally returns many key points around the edges of objects. An illustration of this can be seen in Figure 2.6.

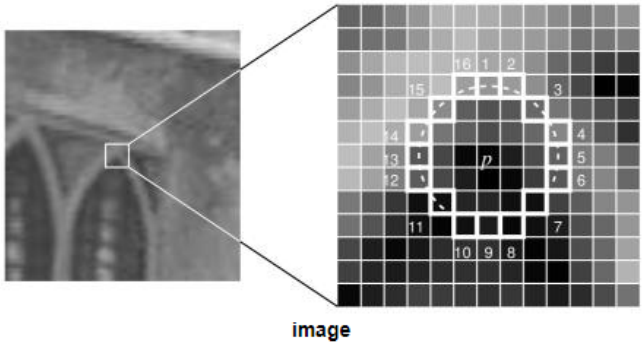


Figure 2.6.: A potential interesting point in an image. In the original paper the circumference of the selected circle is 16 pixels (taken from Rosten 2006 [25])

ORB overview

It is important to know how key points/features are extracted in each frame to understand some of the behavior of the developed Motion Compensation (MC). ORB (Oriented FAST and Rotated-BRIEF) is a feature detector created at OpenCV labs. ORB is in combined FAST keypoint detector with 2.3.2 with a heavily modified BRIEF descriptor [26]. After FAST has found keypoints, Harris corner measure is used to find the best N keypoints. Multi-scale image pyramid is used along side FAST to obtain keypoints at different scales. FAST is unable to compute the orientation of keypoints it retrieves. The orientation of each keypoint is determined by using a intensity centroid. Than a descriptor is calculated based on BREIEF. Normally BRIEF does not work so good with rotations. However ORB guides BRIEF based among others on the extracted keypoint orientations, in what the authors call rotation aware BRIEF (rBRIEF). More about the ORB algorithm can be read in the appendix E.

2.3.3. Descriptor based keypoint matching

There are two main steps in feature matching. The first step is to detect a set of distinct features in an image and compute a local descriptor (feature vectors) for the patch around each of them

(2.3.2). The second step is to match those features by associating feature points extracted from two different images. When matching two sets of descriptors, the distance between these feature vectors is minimized. A commonly used distance measurement is the Hamming distance:

$$d(f_a, f_b) = \sum XOR(f_a, f_b), \tag{2.1}$$

where f_a is a feature descriptor in the first and f_b is one for the second image. The threshold set on the distance measurement will affect the match performance. There exist several possible matching strategies.

The most intuitive strategy is to take one descriptor from the first image and compare it to all the descriptors from the second image, using distance calculation. The closer a distance is, the more likely the feature vectors are matching. However, if several matches are almost as good, they might be ambiguous or incorrect (2.7). David Lowe proposed a filtering method [18] to eliminate features that might cause problems. The two matches with the smallest distance measurement are kept. Using the nearest neighbor ratio test, Lowe determines if the two distances are sufficiently different. The ratio test is as follows:

$$\frac{d(f_a, f_b^1)}{d(f_a, f_b^2)} \tag{2.2}$$

If the distance ratio is low, f_b^1 is a potential good match. On the other hand, a high distance ratio indicates that the distances are not different enough, and the features get discarded.

An alternative to finding reliable matches is the cross-check test. If f_b is the best match for f_a in I_b and f_a is the best match for f_b in I_a, the involved features are kept, else they are discarded.

If there is only a small set of feature descriptors in the images, trying all the possibilities (Brute forcing) will give the best matches. In OpenCV, this is implemented with BFMatching. However, when there are a large number of features, this becomes too computationally expensive. A less expensive alternative is to approximate matches. FLANN (Fast Library for Approximate Nearest Neighbours) builds a data structure (KD-Tree) which is used to find an approximated neighbor. This does not guarantee finding the best matches, but it is much faster. FLANNBasedMatcher is also a part of OpenCV.

Figure 2.7.: The feature in I_a is a incorrect match to the best matching features proposed in I_b (taken from lecture slides by haavardsholm UiO [11])

2.3.4. Estimating transformation parameters from associated keypoints

Here the parameter estimation of a homography transformation matrix is shown. However, a similar approach can also be used to estimate an affine transformation matrix.

First, a point correspondence between key points in image A, u_i and key points in image B, $\hat{u}_i$ need to be established (e.g., the corresponding points in figure 2.5). This can be done by estimating the optical flow. However, when there is such a big distance change between the different points like in figure 2.5, feature-based matching is often more suitable. The key points are extracted separately from each of the two images. A descriptor is generated for each of the key points. The point correspondence $u_i \leftrightarrow \hat{u}_i$ is than determined by matching the descriptors. It is expected that some correspondences might be wrong (e.g., see figure 2.7). A transformed point $\hat{u}_i$ can be found with the following expression, $\mathbf{H}u_i = \hat{u}_i$.

The h_9 entry is usually 1 in a homography matrix. This gives a homography 8 degrees of freedom. However, it is normal to assume that all the nine elements in the matrix are unknown. This is because there are some special cases where the h_9 entry becomes 0 in the solution. First set up the $\mathbf{H}u = \hat{u}$ equation.

$$\begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \hat{u} \\ \hat{v} \\ 1 \end{bmatrix}$$

In this case, the $\mathbf{H}$ and u are known. However, when $\mathbf{H}$ is the unknown parameter out of the three, it needs to be estimated. The above matrix equation can also be written as a set of 3 equations:

$$\left\{ \begin{array}{c} uh_1 + vh_2h_3 = \hat{u} \\ uh_4 + vh_5 + h_6 = \hat{v} \\ uh_7 + vh_8 + h_9 = 1 \end{array} \right\}$$

This relationship between the parameters can then be expressed in another way:

Let $\mathbf{A}$ be the following matrix:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & -u & -v & -1 & \hat{v}u & \hat{v}v & \hat{v} \\ u & v & 1 & 0 & 0 & 0 & -\hat{u}u & -\hat{u}v & -\hat{u} \\ -\hat{v}u & -\hat{v}v & -\hat{v} & \hat{u}u & \hat{u}v & \hat{u} & 0 & 0 & 0 \end{bmatrix}$$

and let h be the following vector:

$$\vec{h} = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{bmatrix}^T$$

Than:

$$\mathbf{A}h = 0$$

The last row in $\mathbf{A}$ is a linear combination of the first two rows:

$$row_3 = -\hat{u} \cdot row_1 - \hat{v} \cdot row_2$$

This means that every $u_i \leftrightarrow \hat{u}_i$ correspondences is contributing with 2 equations to find the 9 unknown entries in h. All n-corresponding keypoints can be used to build a matrix $\mathbf{A}$ of dimension $2n \times 9$:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & -u_1 & -v_1 & -1 & \hat{v}_1 u_1 & \hat{v}_1 v_1 & \hat{v}_1 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -\hat{u}_1 u_1 & -\hat{u}_1 v_1 & -\hat{u}_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -u_n & -v_n & -1 & \hat{v}_n u_n & \hat{v}_n v_n & \hat{v}_n \\ u_n & v_n & 1 & 0 & 0 & 0 & -\hat{u}_n u_n & -\hat{u}_n v_n & -\hat{u}_n \end{bmatrix}$$

There is the constraint that h has to be homogenous; therefore, the matrix $\mathbf{U}$ only needs to have rank 8. From this, it follows that only 4 points corresponding to each other are needed, given that 3 of them are not lying on the same line (are collinear).

Direct Linear Transformation (DLT) is used to construct the homography matrix:

1. First matrix $\mathbf{A}$ is build using at least 4 random pairs of corresponding points, $(u_i, v_i) \leftrightarrow (\hat{u}_i, \hat{v}_i)$.
2. The non-trivial solution of $\mathbf{A}h = 0$ is calculated using singular value decomposition (SVD). $\mathbf{A}$ gets decomposed into USV^T (see D for the theorem).
3. $\mathbf{S}$ and $\mathbf{V}$ are two of the three decomposed matrices. If it turns out that $\mathbf{S}$ is diagonal and all the values on the diagonal are positive in descending order, then the vector h equals the last column of $\mathbf{V}$.
4. The 3×3 homography matrix $\mathbf{H}$ is then reconstructed from the 9×1 vector $\vec{h}$.

In practice, the DLT algorithm is limited to only use four corresponding point pairs for better performance. A normalization and demoralization step is often included in the DLT algorithm to ensure that all the terms of $\mathbf{A}$ have a similar scale.

However, now that a basic homography matrix is estimated for 4-corresponding points, a set of inliers need to be determined. Given an estimated homography $\mathbf{H}_{est}$, and a set of corresponding points $(u_i, v_i) \leftrightarrow (\hat{u}_i, \hat{v}_i)$, S_{cor}. When applying $\mathbf{H}_{est}$ to (u_i, v_i) a estimated set of corresponding points S_{est} can be found. An entry is only added to S_{est} if the error between the estimation and a corresponding entry in S_{cor} is lower than a threshold, ϵ. This is also called the projection error, a geometric error, which is the distance between a projected point (in this case point (u_i, v_i) is projected using $\mathbf{H}_{est}$) and a measured point (in this case $(\hat{u}_i, \hat{v}_i)$). At the end, if the length of the set S_{est} is larger than the length of the set of inliers, S_{in}, $S_{in} = S_{est}$ and $\mathbf{H} = \mathbf{H}_{est}$. Using the set of inliers S_{in}, $\mathbf{H}$ can be re-estimated using DLT.

These are the steps in RANSAC to get a robust homography estimation. When estimating an affine transformation matrix, only six instead of nine parameters need to be estimated. This means that the minimum number of corresponding points needed is only three instead of four (2 of the 3 points must not be collinear). [20]

RANSAC

RANdom SAmple Consensus algorithm is an approach to estimate parameters of a model from input data that has a large amount of outliers. For instance the underlying parameters of a homography transformation matrix or an affine transformation matrix. Since some of the matching points are expected to be wrong (2.3.4), there are a number of outliers.

Other common estimation techniques such as M-estimators and least-mean originate from the field of statistics. RANSAC, on the other hand, was developed within the computer vision community, as a propose by Fischler and Bolles [8]. It is often used in feature-based matching to find a transformation

that best transforms one image to a second image, given a set of matching feature points in both images.

RANSAC resamples and generates possible solutions to estimate the underlying parameters of a model. A common technique other algorithms use is to obtain a starting solution with as much data as possible and then prune the outliers away. RANSAC is a more button-up approach. It starts with the smallest set of input data needed to estimate the model. The aim is to make this set bigger by including consistent data.

A summarization of the algorithm [5]:

1. Randomly selects a minimum number of data points that later will be used to find some parameters of the underlying model.
2. Use those points to find the underlying parameters of the model
3. For all those points, find the number of points that fit a given tolerance ϵ and add them to the set of inliers.
4. Calculate the fraction of a number of data points that are inliers over the number of all the points. If this fraction is higher than a given threshold, the model parameters are re-estimated using the set of all the inliers found before the algorithm is terminated.
5. If the threshold is lower, the above steps 1 to 4 are repeated.

How many times it gets repeated depends on a predefined number of iterations N. To make sure that at least once when choosing a random sample set, it does not contain any outlier, N is chosen to be so high that the probability for that to happen is p (often $p = 0.99$).

Let u be the probability that all the points randomly selected in one iteration are inliers and $v = 1 - u$ that one of them is an outlier. When sampling m random numbers, the minimum number of iterations N required is.

$$1 - p = (1 - u^m)^N$$

solved for N:

$$N = \frac{log(1-p)}{log(1-(1-v)^m)}$$

3. Method

In this chapter, the developed motion compensation (MC) is presented, along with the three modules that build upon it. The post-processing module is about post-processing the output from an already existing detector. The Multi-Image detector (MID) module, explores the idea of using multiple frames as input to develop a new detector. The Tracking module, is based on the MC to track a detected crack over subsequent frames.

3.1. A brief overview of the pipeline

The overall pipeline that ties all the components together starts with a stack of n frames as input. A stack contains all the n most recent frames, and is in many ways the *time window* on which the pipeline operates. The deeper the stack is, the further back in time the pipeline can retrieve information from. The stack is implemented as a First In, First Out (FIFO) queue of max length n. When a new frame arrives from the video stream it is put into the queue. The oldest frame in the stack is then discarded, if the queue is full. Each element in the queue does not contain a frame directly but rather a frame object. When a new frame arrives a new frame object is initialized. This frame object contains, among others, the frame itself, its associated detection, tracks, and transformation matrix. The detection map and tracker are initialized with place holder values, while the transformation matrix is initialized with the identity matrix. The items get updated along the pipeline. Figure 3.1 shows the items the most recent frame object contains after the stack has traversed the entire pipeline.

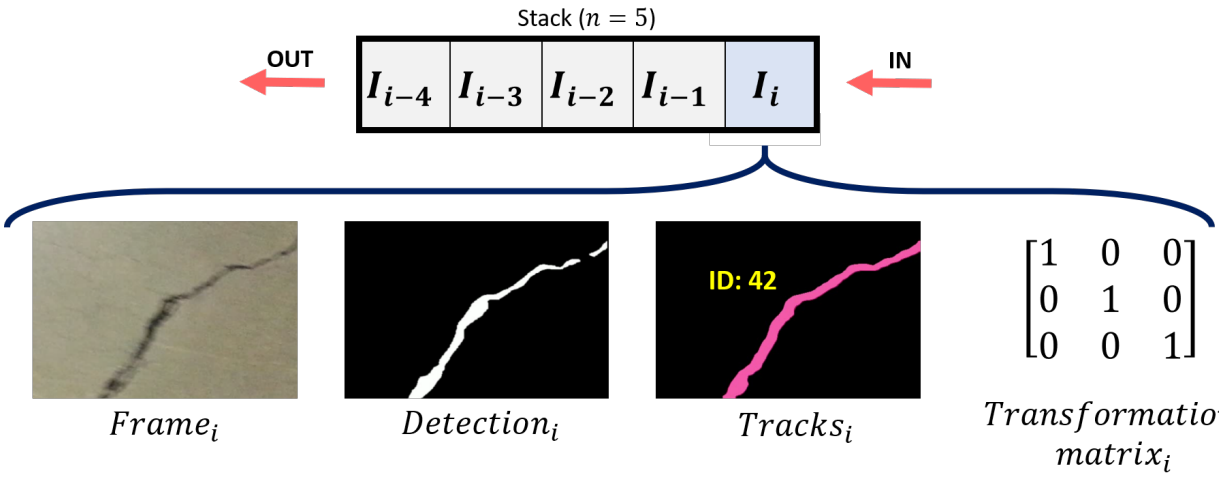


Figure 3.1.: The most recent frame object (I_i) in the FIFO queue, after it has gone through the pipeline

First all the previous frames are motion compensated to the most recent frame. The transformation matrix of the previous frame objects is updated. When applying a previous frames transformation

matrix to itself, the resulting frame is aligned with the most recent frame. That's where the pixel-to-pixel association happens. The same transformation matrix can then also be applied to the binary detection map and the tracks to update their pixel-to-pixel association with the respective items in the most recent frame.

The now motion compensated stack is then put into the MID module, the Multi-Image detector (MID) or an existing Single-Image detector (SID). If the existing detector or the MID do not expect an already motion compensated input, the motion compensation needs to happen after and not before that step in the pipeline.

The results from the detector are now fed into the post-processing module. Using the aligned detection binary maps the detections are processed to output more accurate and temporally stable binary maps. Last but not least the post-processed binary map is sent into the tracking module, where detected cracks are instantiated and associated to previous crack instances. Figure 3.2 gives an overview over the pipeline. As this pipeline is modular, any modules can be replaced or taken out if they are not needed. Keep in mind that the placement of the motion compensation component depends on whether the used detector expects a motion compensated stack or not. It does not matter if the motion compensation is placed before or after a detector, if the detector only operates on single images. This is because the motion compensation only updates the transformation matrix of previous frames and not the most recent frame, which is the only frame a SID looks at.

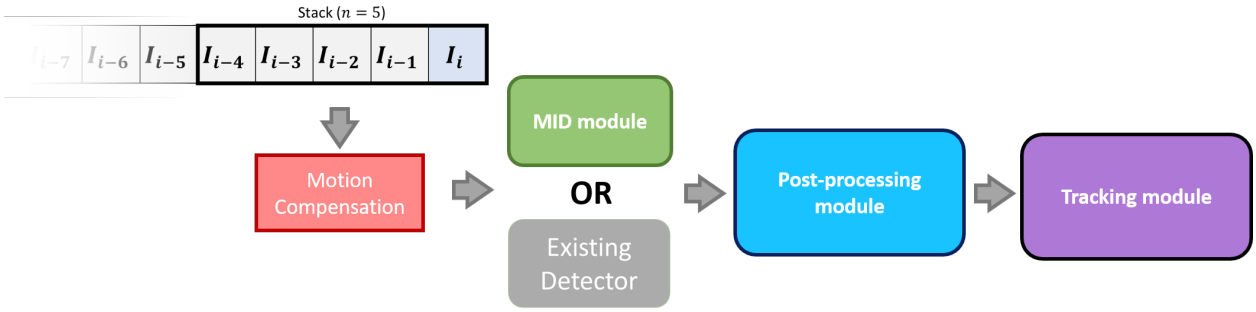


Figure 3.2.: An overview of the developed object detection pipeline developed in this project. Here the stack has a depth of $n = 5$

3.2. Motion Compensation

Semantic image segmentation is about giving each pixel in an image a class label. There are many different benchmarks to measure the performance of semantic segmentation models [15] [7] [27]. However, those models are mainly designed to be used on single images. When applying these models on a video sequence, every frame in that sequence is looked at individually. Frames in a video often correlate to each other, and spatial-temporal information can get lost if the context of the previous frames is not considered.

Given a video stream, Motion Compensation (MC) between previous frames and the current frame can be used to get a pixel-to-pixel association between them, making some of this lost spatial-temporal information more available to be recovered. To do this, a transformation between the previous frame and the current frame needs to be found.

Instead of dividing a frame into blocks, a couple of features are extracted from the entire frame. A transformation matrix can be estimated by matching the features from a previous frame to the current frame. The transformation matrix describes how each pixel in the previous frame is remapped to form a transformed image. This transformed image should match the current image

as much as possible. Note that the transformations estimated are 2D transformations and, as such, are only expected to be viable in planar scenes. It is assumed that the only movement in-between frames originate from the camera moving in space. The feature/keypoint extraction in this project is done with ORB (2.3.2).

There exist different methods to match extracted features between two frames with each other. Differential-based methods (i.e., optical flow estimation) try to determine what moved and how it moved. However, these types of estimations often struggle with large movements in-between frames. [1]. Therefore this project uses descriptor-based matching (2.3.3), which handles larger movements better. More specifically, the descriptor matching method *FLANN* is used as it has good performance and comes shipped with OpenCV.

When estimating a 2D transformation matrix, four different transformation types are tried out: *translation*, *similarity*, *affine* and *homography*. Each of the transformed frames is then evaluated to see how good a match they are to the current frame. The transformation with the best score is then used. While this makes the motion compensation overall more robust, it comes with some computational overhead and is, therefore, a little slower.

The *Peak signal-to-noise ratio* (PSNR) metric is calculated between the grayscale versions of the compensated frame and the current frame. This is to evaluate how successful a transformation has been. The PSNR value is based on the logarithm of the *Mean Square Error* (MSE) between two images. The MSE is calculated by using the $M \times O$ dimension of the grayscale frames.

$$MSE = \frac{1}{M \times N} \sum_{x=1}^{M} \sum_{y=1}^{N} [(I_{x,y} - I'_{x,y})^2] \tag{3.1}$$

Where M and N are image resolution, $I_{x,y}$ is the pixel value in the current frame at the coordinates x, y. I' is the motion compensated frame. The PSNR value is then given as:

$$PSNR = 10 \log_{10}(\frac{\max^2}{MSE}) \tag{3.2}$$

Where max is the maximum possible pixel value (in an 8-bits grayscale image, the max pixel value is 255). The results are measured in decibel (dB). The lower the PSNR value is, the less similar the frames are. If the two frames are identical, the PSNR value is infinity or undefined ($MSE = 0$, division by zero).

The PSNR metric is, as the idea of the motion compensation, borrowed from the field of compression. Here it is mainly used to measure the quality of a reconstructed image from lossy compression codecs. In the case of MC, it is used to compare a transformed previous frame to the current frame. The higher the PSNR value is, the better the MC is. A low PSNR, on the other hand, indicates a bad MC. Therefore a PSNR threshold needs to be set. If the PSNR value of a compensated frame is under the set threshold, the compensation is considered faulty. In this case, the MC is not viable, and all operations based on it need to be skipped. There are several reasons why the MC might fail. Too much movement can make that the previous frame and the current frame share to few features. This can be the case if the camera moves great distances between frames, resulting in not enough features from a previous frame be represented in the current frame. A similar effect occurs when there are smooth surfaces with little details. Then it can become hard to extract good feature points. The more good overlapping features are extracted, the more viable a transformation matrix

[1] In the Specialisation Project Report (Eichsteller 2021), differential and feature-based matching were tested. Overall feature-based matching seemed to be more robust when it came to handling rapid movements.

can be estimated. Motion blur is also a factor that smooths out the appearance of a surface, making it challenging to extract overlapping features. The implementation of the MC assumes that the scenes are planar, which is not always the case. For non-planar scenes, a 3D transformation would be more reliable. However, this is more computationally expensive and difficult to implement but can be considered in future work.

The motion compensation is not only applied to the previous frame but all the $n-1$ previous frames in a stack. Creating such a stack of motion-compensated frames allows for even more spatial-temporal information to be extracted. The naïve approach creating such a stack is to calculate a new transformation matrix for each of the $n-1$ previous frames every time a new frame is added into the stack. However, this can lead to inconsistencies between the previous frames themselves. I.e., the motion compensation between $n-k$ and $n-(k-1)$ can vary when a new frame is added to the stack.

There are other, more clever methods to create a stack of n motion-compensated frames. When a new transformation matrix between the previous frame and the current frame is estimated, it can be used to update the transformation matrix for each of the $n-1$ previous frames as described in 3.2.1. This fixes the problem of inconstancy in the motion compensation among the previous frames themselves. Additionally, it comes with much less overhead since only one new transformation matrix needs to be estimated. The further back in the stack a frame is, the further back in time it is. The oldest frames in the stack can often have moved so much that there are few or no features that overlap with the current frame anymore. Suppose only a few features overlap between the older frames and the current frame (which is likely because of camera movement). In that case, the naïve approach will struggle to estimate a good transformation. With the transformation matrix update approach, this is not a problem.

Because transformation matrices are estimates, they are not expected to give 100% accurate results. The reasons for the inaccuracy is the same reasons why an estimation might fail. In the approach of updating previous transformation matrices, these inaccuracies accumulate. Also, in the näive approach, errors tend to be bigger the further back in the stack a frame is. This is the main reason why a stack needs to be shallow. While a bigger stack would contain more spatial-temporal information, the accumulated errors will corrupt its value.

3.2.1. Equation for updating transformation matrices in a stack

This subsection shows the math behind the update approach. For each pair of images I_n, I_{n-1}, there exist a point-to-point relationship which can be written as

$$\vec{x}_n = \mathbf{T}_{n,n-1} \cdot \vec{x}_{n-1}$$

where the vectors are homogeneous vectors (2.3.1).

$$\vec{x}_{n-1} = \mathbf{T}_{n-1,n-2} \cdot \vec{x}_{n-2}$$

It follows from induction that

$$\vec{x}_n = \mathbf{T}_{n,n-1} \cdot \mathbf{T}_{n-1,n-2}\vec{x}_{n-2}$$

and

$$\vec{x}_n = \mathbf{T}_{n,n-1} \cdot \mathbf{T}_{n-1,n-2} \dots \mathbf{T}_{n-k+1,n-k}\vec{x}_{n-k}$$

This means that the transformation between I_n and I_{n-k} can be expressed as

$$\mathbf{T}_{n,n-k} = \mathbf{T}_{n,n-1} \cdot \mathbf{T}_{n-1,n-2} \dots \mathbf{T}_{n-k+1,n-k}$$

Keep in mind that the order of the dot-product is essential. With all the transformation matrices, the images $I_{n-1}, ..., I_{n-k}$ can be warped to fit I_n. If all these matrices are available, warping image I_{n-k} onto image I_n becomes easy.

3.3. The post-processing module

Given an arbitrary detector that is run over a video sequence, the prediction results often vary between frames. The prediction for each frame may contain some False Positive (FP) and false negatives False Negative (FN). When looking at the prediction results from previous frames, more information can be made available to the current prediction. In the post-processing module, the main focus is to use the predicted label maps from previous frames to post-process the prediction from an arbitrary detector. The arbitrary detector, in this case, is assumed to output a prediction label map, where each pixel is classified to belong to a specific class (pixel-wise predictions for the image). However, this approach could also be extended to work with other detectors, for instance detectors which output bounding boxes. Figure 3.3 shows an example of the output of an arbitrary image segmentation model on $n = 4$ consecutive frames, where there is some camera movement between the frames.

When simply superimposing the prediction in such a stack of n subsequent frames (see Figure 3.4), it becomes apparent that the post-processing step needs to be able to associate each detected pixel in the current frame, to each corresponding pixel in the previous frames, in order to extract useful information.

This is where motion compensation (MC), as described in 3.2, comes into play. The same compensation used to motion compensate the $n-1$ previous frames to the current frame can be applied to each of the frames predicted label maps. A pixel-to-pixel association is thereby established between the pixels in the predicted label maps. Figure 3.5 shows an example of those compensated label maps. Since the post-processing is based on the MC, the assumptions made for a video sequence when using MC also apply for the post-processing step.

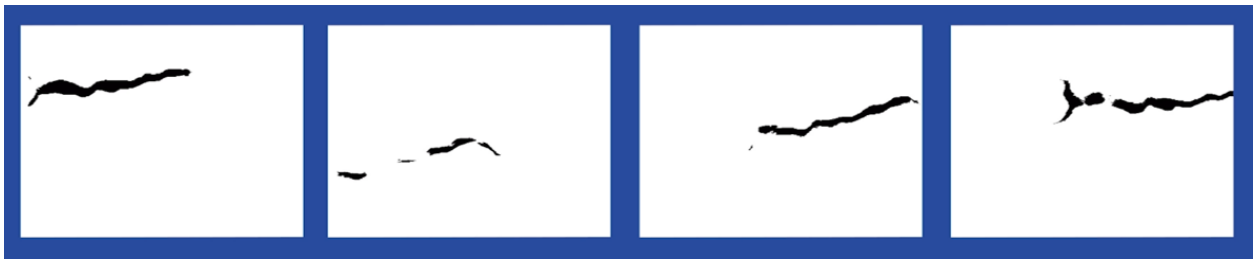


Figure 3.3.: An example of the predicted label maps outputted by an arbitrary crack detector. The detector is run over a 4 frame video sequence, from right to left, making the label map most to the left the most recent. Each black pixel corresponds to a crack in the input frame. Note that there is some camera movement between the frames and that this detector is inconsistent with its prediction

Figure 3.4.: The superimposed stack from Figure 3.3

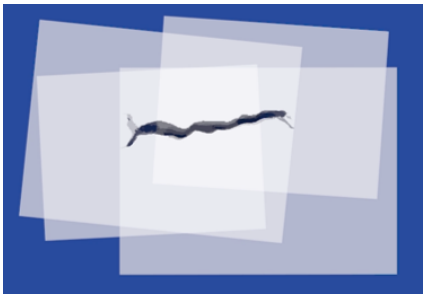


Figure 3.5.: Compensated prediction label maps from the labels from Figure 3.3

This stack of compensated label maps can now be analyzed to make the most recent prediction better. If a detector outputs a FN pixel in the most recent detection, but this pixel is correctly classified in earlier frames, it can be recovered. Likewise, previously predicted label maps can also help to determine if a detected pixel is a FP. In general, when this post-processing step is added to the detection pipeline, it smooths out the prediction results and makes them overall more temporally stable. In this project, the analyses of this stack is implemented in two different ways. One being a pixel-level voting, the other a neural network approach. Both are trying to leverage the available information which is already extracted by a detector in previous frames. They do this by operating on the compensated stack of predicted label maps.

3.3.1. Post-processing with voting

If a FN pixel prediction in the most recent frame has never been detected in previous frames, there is not much that can be done to recover it. However, like seen in Figure 3.3, a detector can often be inconsistent when it comes to detecting a pixel as a TP over multiple frames. This results in a temporally flickering of the detection. If the detector would never output any FP, the lost TP in the current frame could easily be recovered by superimposing the compensated detection results, like in Figure 3.5. The flickering of the detection would then be smoothed out. Unfortunately, a detector can also have FP results, and this approach would boost FP detections to become even more present in the current prediction result. If a FP detection is consistent over multiple frames, it is indistinguishable from a TP for the post-processing step. However, it is often observed in detection results from Single-Image detectors (SIDs) that many FP detections are short-lived. They show up in one or two frames but are not seen again in the subsequent frames. The reason for this temporally detection instability can be primarily be traced back to the nuances in the appearance of the pixels over subsequent frames. This behavior of some FP can be used to filter them out. Detected pixels that show up in the stack over a threshold number of times are assumed to be TP. From now on, this is referred to as the *Voting approach*.

In Figure 3.5, the threshold is set to $\lambda = 2$. Given that a pixel is black in 2 out of 4 frames in the stack, it is outputted as a positive detection. If a pixel is black in fewer frames, it is assumed to be a FP, and is therefore outputted as a negative detection. The post-processing is only done for the class "*cracks*" in this project, but it can theoretically also be expanded to work with more classes. Note that this is an illustrated example. In the real prediction binary label maps the colors are the opposite (white pixel = crack, black pixel= no crack),

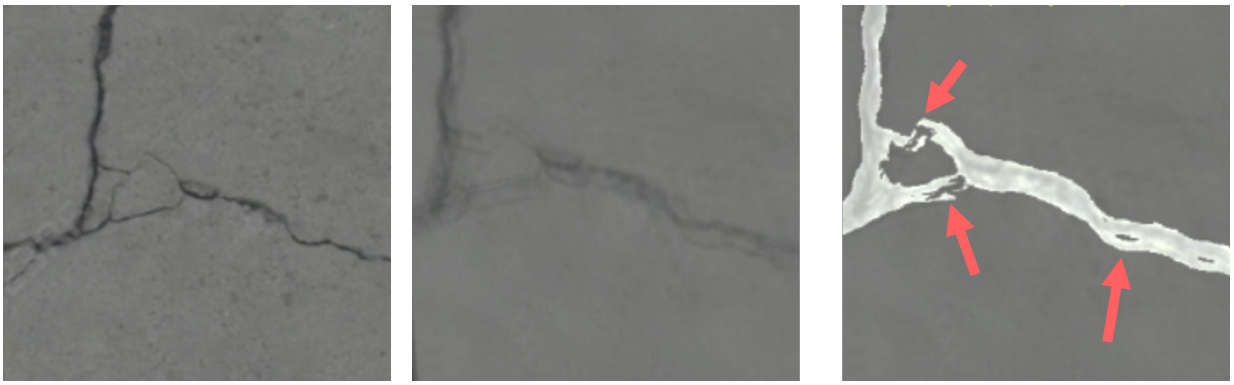
Each pixel in a stack is voted on. For a pixel to be classified as a crack, it needs to be detected as such in at least λ out of n subsequent frames. What this threshold λ should be set to depends on multiple factors. As discussed in 3.2, the MC is less accurate the further back in the stack a frame is. Therefore the total stack depth must be limited. While a deeper stack contains more information, this information is useless if the MC is bad. That is also why a stack needs to be reset if the MC is not performing well enough. In this project, the performance of the MC is measured with the PSNR between frames. When a stack is empty, the majority voting has nothing to work with. The input of the majority voting should therefore also be its output. Either λ is set to a percentage, or it is set to a number. In the latter case, λ might need to be adjusted continuously, depending on how many frames are available in the stack. Alternatively, a minimum number of frames that need to be present in a stack can be required before the post-processing kicks in.

When the camera moves in the video sequence, new objects might appear in the scene. If the detector detects these new appearing objects, there will be a delay of λ frames before the voting approach accepts the detection as valid. The bigger the λ, the longer the delay. Therefore, the λ value needs to be set so that it takes the trade-off between how sensitive the voting approach is and how long

the delay is into consideration.

3.3.2. Post-processing with Neural networks

temporally stable results. However, it assumes that consecutive frames are perfectly motion-compensated. This is an assumption that not always will hold up. There are multiple reasons why the motion compensation between frames might be inconsistent (see 3.2). This results in minor offsets in between the motion-compensated frames, which visually looks like a "ghosting" effect, like in Figure 3.6. The same "ghosting" effect will also be present in the binary detection mask. Classified pixels in the masks that are not fully overlaying each other will be ignored by the majority voting. By replacing the majority voting with a Neural Network, this spatial awareness can be learned. The Network needs to learn to filter out false positives and false negatives, similar to the majority voting. It also needs to learn to compensate for minor errors in the motion compensation.



(a) Stack of $n = 5$ successful MC frames

(b) Stack of $n = 5$ unsuccessful MC frames, resulting in offsets, and a ghosting effect. Voting on bad MC detections (see right image), can lead to faulty results (red arrows point at some)

Figure 3.6.: Ghosting effect when MC is inaccurate. Note that this is an extreme example to visualize the ghosting effect; often this effect is much more subtle

Network architectures

Like the majority voting, the Network needs to operate on a stack of n binary prediction label maps. It is ensured that the Network will look at the stack of binary prediction masks as a whole when using convolutions in the input layers. Overall this will make the Network capable of becoming more spatially aware. The output is, as with the majority voting, a single binary label map.

Conventional Convolutional Neural Networks (CNN) have convolutional layers followed by a couple of fully connected layers (FC). FC layers always expect a certain input size. By avoiding FC layers and using an FCN architecture instead, images of any size can be used as input. In general, FC layers lead to a loss in spatial information, so they are usually only reliable for segmentation on small restricted areas. Since spatial information is vital for the post-processing Neural Network, the FCN architecture is chosen. Note that such a network expects a fixed amount of input channels, which in this case refers to the number of label maps in a stack. This means that if the stack contains fewer label maps than the required number of input channels, the existing label maps in the stack need to be duplicated to fill it up. In this implementation, only the most recent detection map is copied to fill up the stack. This is done to keep things simple.

Networks used for image segmentation, such as UNet, utilize an encoder-decoder architecture with skip connections. This kind of network style is promising since the input and output dimensions are similar to what the post-processing neural networks need. Additionally, these architectures have a heavy emphasis on learning spatial information. Other architectures, like the GoogleDeepLabv3 architecture (which utilizes atrous convolutions), can be retrofitted to this task as well.

A total of 4 FCN architectures are developed/modified to become post-processing neural networks:

1. Small Network (*SmlNN*) [8 layers]
2. Small Network with skip connections (*SmlSkipNN*) [8 layers]
3. UNet based architecture[2] (*UNetNN*) [23 layers]
4. Google DeepLabv3 (ResNet101 backbone) based architecture[3] (*DLv3NN*) [101+ layers]

An overview over the architectures can be found in the Appendix A. The two smaller Networks are built from the ground up in this project. The SmlNN consists of 4 convolutions in the encoder part and 4 transposed convolutions in the decoder part. The skip connections introduced in the second Network are inspired by the skip connections used in UNet. The SmlSkipNN is, in many ways, simply a UNet architecture with a total of only 8 layers. Instead of transposed convolutions, the decoder part uses bi-linear upsampling. This is why the SmlSkipNN has the least amount of parameters of all NNs in the post-processing module.

Both UNet and DeepLabv3 are usually used for image segmentation tasks and are restricted to a 3 channel input (RGB). The number of input channels is altered to be n (in this project $n = 5$). Each of the channels represents a compensated detection map in the stack. As the expected output is the same as for image segmentation, the loss function remains unchanged.

3.3.3. Training data and augmentation

This project has access to 4000 labeled crack images from inside ship tanks provided by DNV. The data set is supplemented with around 11 000 labeled images containing cracks in streets and concrete to add even more variances in crack appearances. These labeled images are publicly available and originate from different crack segmentation datasets [33, 32, 6, 28, 1, 34]. The ground truth labels are binary maps that pixel-to-pixel map to the original image. White pixels are associated with crack pixels, while black pixels are associated with no-crack pixels. For training the NNs in the post-processing module, only the ground truth label maps are needed. However, the input of the models is supposed to be a stack of overlaying detection masks, not a single label map. Normally this stack originates from n subsequent motion-compensated frames in a video sequence. However, since no labeled videos are available for this project[4], a stack of binary detection masks is emulated as a part of the augmentation process.

Augmentation of training data can be used to expand the training data set. By providing more variation, the model will learn to generalize better and achieve better overall performance. All the augmentations for the data in the dataset for the post-processing NNs, are done "on the fly". The augmentation of the data is not done in a pre-processing step but rather during training. This way, the Network will rarely receive identical-looking stacks of label maps during training. The more epochs are run, the more unique data the model is exposed to.

[2] https://github.com/milesial/Pytorch-UNet
[3] https://github.com/pytorch/vision
[4] some synthetic video generation is introduced later in the MID module (3.4.1)

When a ground truth detection mask arrives at the augmentation step, it firstly gets copied into n additional copies, which will emulate the compensated stack of label maps. Each of those copies is then randomly circular shifted with a maximum offset of δ-px. Circular shifting means that an image gets translated. The parts of the image that get outside the frame window appear again on the opposite side. These slight offsets are there to emulate errors that the motion-compensation might have. In retrospect, normal shift would probably also have been sufficient. All the ground truth label maps in a stack are randomly flipped (all are flipped the same way).

Some of the following augmentations use perlin noise[5], which is a type of gradient noise, to procedural alter the training data. Generating perlin noise has some overhead. Instead of generating perlin noise during training, many big perlin noise-maps ($2048px \times 2048px$) are pre-generated before training. During training, a random pre-generated perlin noise map is loaded. A random crop is applied to fit the size of the noise to the size of the ground truth mask before using it to apply an augmentation.

Different augmentations might need perlin noises with different kinds of "resolution". An example of different resolutions in perlin noises can be seen in Figure 3.7. The higher the resolution is, the more fine-grained the noise is. By pre-generating around 1000 perlin noise maps for each augmentation type that needs one, combined with the random cropping, ensures a large amount of randomness during augmentation.

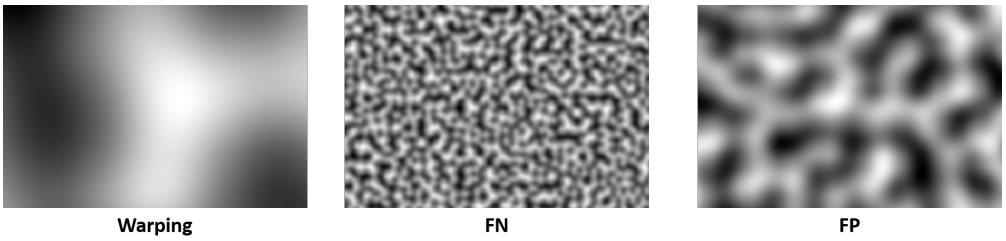


Figure 3.7.: Different resolutions are needed for the different kinds of augmentations

Warping

The crack images can be warped slightly as long as the warping is applied equally on the whole emulated compensated stack. This is to get even more variations in the available training data. Warping is done by taking a vector field and displace the pixels in an image accordingly. The same warping vectors need to be applied throughout the entire compensated stack of label maps.

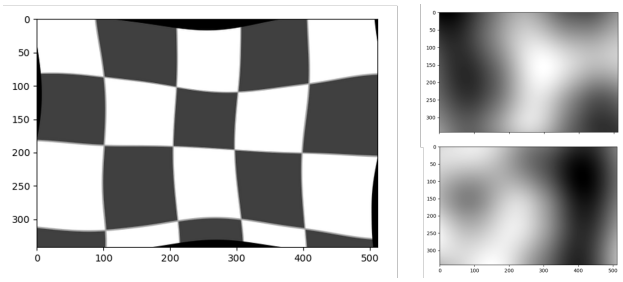


Figure 3.8.: To the left, a warped image (the original image has just a straight check pattern). To the right, the two perlin noises that are interpolated in-between to obtain the vector field used for warping. The perlin noise images have the same size as the image that is to be warped.

[5]https://github.com/pvigier/perlin-numpy

A vector field is obtained by generating two perlin noise images and then interpolate between the two (see Figure 3.8). Obtaining the vector field is done with *np.meshgrid()*[6], and an image is warped, according to the vector field, with *cv2.remap()*[7].

Generate False negatives

A False Negative (FN) detections is when a pixel that belongs to a crack is not detected as such. This can be a temporally inconsistency a crack detector might have when detecting cracks in subsequent frames. The following describes how such FPs are emulated.

Given a stack of n-binary detection masks, random gaps in the detections are created in each binary crack mask in the stack. This is done by generating a perlin noise and setting a pixel intensity threshold to make the noise completely binary. The thresholds used in these augmentations are randomly set so that that $20\% - 80\%$ of the pixels in the resulting binary map are "white" (intensity value 1). This is achieved by creating 1000 bins and group together all the different pixel values that belong together in the same bin. When iterating over the bins containing pixels from the highest to the lowest, all the containing pixels of each bin are set to white, up until the desired white to black ratio is reached. The resulting binary noise map is then subtracted from the binary mask and clipped back to the values $\{0, 1\}$. The process of combining this perlin-noise-based binary map and a ground truth binary map is shown in Figure 3.9 and 3.10.

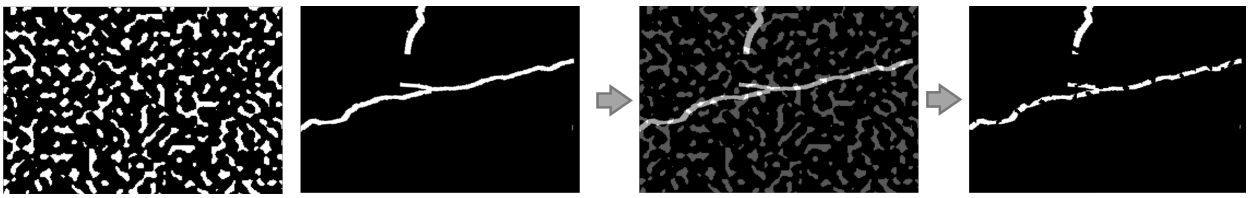


Figure 3.9.: The binary map resulting from setting a pixel intensity threshold is combined with a ground truth label. All the "white" pixels that overlap in the two binary maps are removed (set to 0). This creates random gaps in the ground truth cracks

.

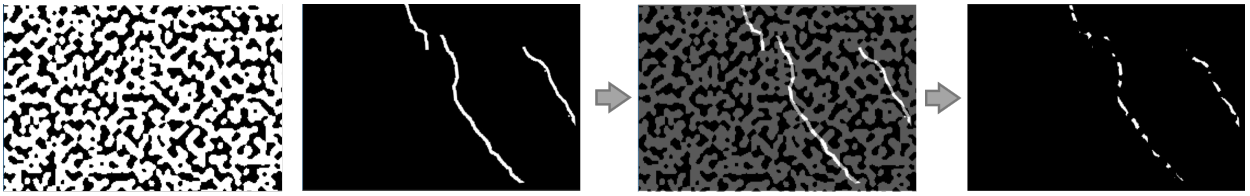


Figure 3.10.: Similar as in Figure 3.9, but a higher pixel intensity threshold, creates bigger gaps in between else connected crack pixels.

Generate False positives

A crack detector can also have some False Positive detections. This is when a pixel that is not a crack is classified as a crack. If a pixel has the non-temporary appearance of a crack, this FP might be consistent in-between frames. When only looking at the binary detection masks, it is practically impossible to know if the detection is a FP. However, often there are FP that only occur in a few

[6] https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html
[7] https://docs.opencv.org/3.4/d1/da0/tutorial_remap.html

detections, but not in all of them. The voting removes all the FP that are only detected in less than λ frames. The NNs need to learn to do something similar.

To generate FP, a random binary ground truth mask is chosen from the training dataset. Perlin noise is used in the same way as when it was applied to generating the FN (see 3.3.3). This time the perlin noise has a much lower period, meaning a lot more of the white pixels from the ground truth masks will be removed. This process is done twice, with different perlin noise maps to remove even more pixels at random.

The resulting mask is then random circular shifted and flipped before added together with a binary-detection mask and clipped to the binary values $\{0, 1\}$. Figure 3.11 shows the generation of a FP binary map, that is used to generate FP on a GT binary map in Figure 3.12.

Note that this process is done independently on all the GT masks in a stack. This approach, therefore, only emulates FP detection results that occur once in a stack of frames.

When given a stack of n-binary detection masks, this process is done separately for each element in the stack. This augmentation of generating FP comes after the augmentation of creating FN. Which order they are in is not particularly important, though it would result in slightly different augmentation outputs.

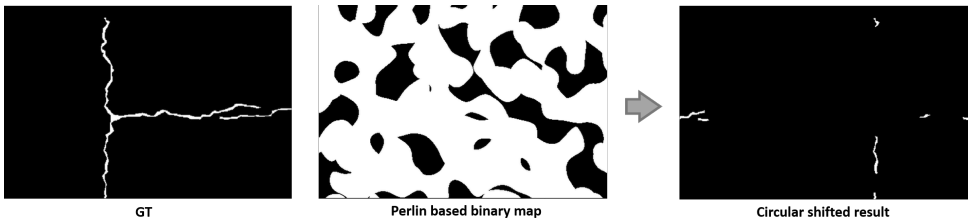


Figure 3.11.: Like in False negative generation, a perlin based binary map is used to subtract GT pixels. The resulting binary map is additionally circular shifted. The white binary pixels represent FP detection results

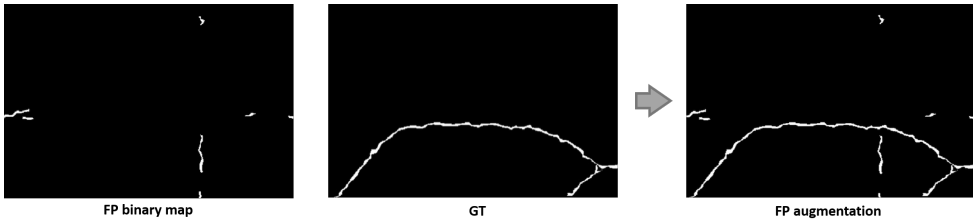


Figure 3.12.: The resulting map from Figure 3.11 is combined with a GT to generate FP detection results

3.4. The Multi-Image Detector module

When inference is run on a video sequence with a single image detector (SID), the detector will look at one frame at a time without considering the previous frames. The appearance of a crack can vary a lot between frames because of, i.e., changing lighting conditions, compression artifacts, motion blur, and changing shadows. The difference in appearances of the target object is one of the main reasons why SIDs tend to be inconsistent with their detection from one frame to another.

The post-processing module is about using MC for post-processing the output from an input detector. The stack of motion-compensated detections enables the NNs in the post-processing module and the voting approach to combine previously extracted information with the current detection. The processed information consists of binary detection maps, which are deeply dependent on the detections done by the underlying detector. These binary maps do not directly contain information about a crack's change in appearance. If the detector did not detect a crack that changed its appearance from one frame to another, information is lost.

Training a detector that not only looks at the current RGB frame but also takes previous frames into account will make the detection more robust to variation in crack appearance. In this project, this is referred to as a Multi-Image detector (MID). Figure 3.13 shows an example of a number of subsequent frames. It is evident that there are differences in the frames, i.e., how the light reflects and motion blur. A SID only operates on the most recent frame, in this case, $frame_{(i)}$. As the most recent frame has some intense light reflection, extracting all the same information as in the previous frames becomes hard. Those sudden appearance variations are the main reasons for the temporally instability SID detectors often have. A MID can extract information from the different appearances in all the $n = 5$ frames, giving it more information to work with. Therefore, MIDs should, in theory, be more temporally stable.

However, there is often motion between frames. It can be beneficial to motion-compensate previous frames to the current frame before feeding them into such a Multi-Image-Detector (MID). The MID can then focus on extracting as much information from the pixels associated with each other and may have variation in appearances. It does not need to focus on the change in the pixels localization (besides compensating for minor MC inaccuracies). The information from previous frames and the current frame are combined together and utilized to make a more educated crack prediction on the current frame.

Figure 3.13.: An example of $n = 5$, subsequent frames in a real life video, where $Frame_{(i)}$ is the most recent frame.

3.4.1. Training data and augmentation

The ideal training data for the MID would be video sequences of cracks with labeled ground truth for each frame. Training a MID with real-life video frames would make it robust to the artifacts and appearance variances in-between frames. Unfortunately, just a handfull of unlabeled video sequences of cracks are available. Early in the project, it was tried to hand label the few available video sequences. It turns out that this is highly time-consuming and would not yield enough variety

in the training data. Labeling tools such as CVAT [8] provide some tracking of the labels in-between frames. However, this is not always accurate, and manually tweaking is still required at each frame. There is also the possibility to only label every i-th frame and then interpolate the labels in-between. This turns out to produce some inaccurate labels as well. At the end of the day, it is difficult not to have minor variations of the ground truth labels in-between frames, making the labels themselves temporally unstable.

The available data, which is introduced in the post-processing module (3.3.3), consists of a total of around 15 000 still crack images and their associated label map. This data can be used to generate synthetic videos with associated ground truth. The ground truth is guaranteed to be temporally stable and accurately associated with each synthetic frame, given that the original ground truth is accurate. Such synthetic videos need to feature the behaviour of a moving camera.

Extracting camera movement from real videos

There are different ways the camera moving effect can be achieved. One way is to generate a camera movement algorithmic. However, creating an algorithm that generates sufficiently realistic results, is complicated. Another option is to extract the camera movement directly from actual real-life video sequences, making the movement more natural.

For this approach, the camera movement is tracked on real-life videos. The videos used are from the dataset of the DAVIS Challenge on Video Object Segmentation 2016 [21]. The movement is extracted using the same process of estimating a 2D transformation matrix between two frames, as in the motion compensation. The associated transformation matrix at any given frame is the dot product between the extracted transformation matrix and ALL the previous transformation matrices (essentially using 3.2.1 with an unlimited stack depth). If the real-life video comes to an end or when calculating a transformation fails, the associated matrices for each time step are returned. The extracted camera movement becomes transferred by applying the associated transformation matrix for each frame to a still image. The same transformation matrices are then applied to the associated ground truth of the still image. This means that the ground truth will be consistent in its appearance and always be accurate, given that the initial ground truth is accurate. In this approach, only one image needs to be labeled to generate an entire synthetic video. While the datasets used are already labeled, it is observed that labeling all the crack pixels in an image is hard. Especially complicated is it to determine where a crack begins and where it ends. Even in the pre-labeled dataset, the labeled ground truth might miss a few crack pixels here and there. When the dataset is huge, those few inaccuracies do not matter. However, it is something to keep in mind when evaluating the trained models on small datasets.

Emulate other video behavior

Some different augmentations are applied on the frames to emulate real-life videos even more. Brightness, contrast, and saturation are altered randomly between frames to emulate changing lighting conditions. In real-life footage, the lower the camera's shutter speed is, the more motion blur there will be in the video, especially if the motions are rapid. Therefore to make the model more robust, motion blur is added randomly in frames. Often video sequences and video streams use lossy compression to save on disk space and transfer time. However, this often results in compression artifacts. Many compression algorithms divide a frame into blocks. The block boundaries can become visible if there is a lot of information change in frames and the bit rate is low. A subtle average pooling is applied randomly to emulate this blocky effect.

[8] https://github.com/openvinotoolkit/cvat

These are emulations of the behavior of real-life video sequences, not simulations. The augmentations are applied randomly and do not follow the physics and behavior of real-world scenes. Synthetic videos are generated from flat 2D images using 2D transformation. However, this means that they do not emulate any 3D structures. In real-life, the scenes are mostly not planar. A lot of those non-planar features get, therefore, lost in the synthetic videos. Features such as the parallax effect, where the background moves slower than the foreground, or how light reflects and interacts with different surfaces. Even cracks themselves, the targeted object, are 3D structures that can change their appearance between frames because of those factors. Additionally, the synthetic videos do not capture how light conditions, motion blur, shadows, and even the compression changes over time.

There will be some thoughts on getting better training data and streamlining the labeling process in the future work chapter 7.4. This includes both semi-automatic support for hand labeling and fully automatic synthetic video generation.

Augmentation "on the fly"

The 2D transformation matrix needed to generate a synthetic video out of a still image can be pre-calculated and saved at each time step. This allows subsequent video frames to be generated on the fly during training and testing. Loading pre-calculated subsequent transformation matrices, the transformations can be applied directly to a single image and its ground truth label. As the multi-image model expects n input frames, there is only a need to generate a n frame long synthetic video from each frame during training. Random n subsequent transformation matrices, originating from a random real-life video, are loaded and applied during training. A transformation matrix is a 3×3 matrix, and loading those matrices and applying them is fast. The benefit of generating the videos on the fly during training is that the model will get exposed to various videos with different appearances and movements. Together with the other augmentations done to emulate real-life video artifacts, the models will be exposed to a wide variety of training data. This should enable the networks to generalize better and be less prone to overfitting.

The pipeline for generating videos on the fly during training is as follows:

1. An crack image and its ground truth arrive
2. Random n subsequent transformation matrices, which were extracted from the DAVIS dataset before training, are chosen.
3. Each transformation matrix is applied to an image-ground truth pair.
4. Video artifacts are applied to each of the resulting individual frames independently. For this the libraries OpenCV[9] and Imgaug[10] are used.
    a) Chance of changing brightness, contrast, saturation, hue
    b) Chance of introducing motion blur in a frame
    c) Chance to introduce pix-elating effect to a frame to emulate video compression artifacts (This is done with Average Pooling).

[9] https://opencv.org/
[10] https://github.com/aleju/imgaug

3.4.2. MID module implementation

The implementation of the MID is based on the image segmentation model DeepLabv3 with ResNet101 backbone, which comes shipped with PyTorch. The input to the MID is a stack of n frames where all older frames are motion compensated to the most recent frame. DeepLabv3 is a Single-Image detector (SID), which means that it expects 3 input channels (RGB) out of the box. As the number of input channels needs to be a fixed number when training the model, the depth of the stack needs to be fixed as well. Therefore the input layer of the MID needs to be modified so it can receive $(n) * 3$ channels (given that each frame in the stack is an RGB image and n is the stack depth). In this case, it means modifying the first convolutional layer in the ResNet101 backbone. Because the Network only needs to predict one class, cracks, the classification head of DeepLabv3 is also modified. The loss is calculated between the output of the Network and the ground truth of the current frame. Only one class is classified. The loss function chosen for this is BCEWithLogitsLoss, which is a combination of sigmoid and Binary Cross entropy.

The training pipeline for a single training image is as follows. The whole pipeline also works in batches.

1. Get an image and its corresponding ground-truth binary label mask from the training dataset
2. Generate a synthetic video sequence with ground truth of length n, as described in 3.4.1
3. Run motion compensation on the generated RGB video. This results in a stack of depth n where the $n-1$ previous frames are motion compensated to the most recent frame. The most recent frame itself remains unchanged, and its corresponding ground truth, from the binary label video, becomes the ground truth for the stack.
4. Forward pass the stack
5. Calculate the loss between the output and the ground truth from the most recent frame, using BCEWithLogitsLoss
6. Do backpropagation.

As the MID is trained on a fixed depth $(n \cdot 3)$, it is crucial that the input, when running inference, always has the same number of input channels. However, there are not always n frames available in a stack. This is the case at the start of a video stream or when the stack recovers from an MC error. In this implementation, the most recent frame is duplicated to fill up the stack in those cases.

A MID that does not use the MC is also trained (MID no MC). The same pipeline applies as before, only without the MC step. The idea is that the MID without the MC learns to motion compensate on its own.

A single image detector (SID) that uses the DeepLabv3 architecture right out of the box is trained as well. The only difference in training the SID is that it only receives one frame in a synthetic video instead of the n subsequent frames. Like in the "MID no MC" the MC step is skipped in the training pipeline. Training a SID and the two MIDs, which all are based on the same architecture, training data, and augmentations, will give a fair comparison between the different approaches. The SID is the underlying detector used for the post-processing module in the experiments. It also acts as the baseline when evaluating the performance of the different approaches in the post-processing module and the MID module module.

3.5. The Tracking module

The MC developed in this project operates on a stack of subsequent frames and provides a pixel-to-pixel association between the most recent frame and all the older frames in the stack. In practice, this means that the MC tracks all the pixels available in the most recent frame backward. If a pixel in the most recent frame exists in an earlier frame in the stack, they are associated with each other, given that the MC is correct. Pixels that belong to a crack are only a subset of all the pixels in each frame. If all the pixels in the most recent frame can be tracked, so can this subset of pixels. It is important to note that this type of object tracking only works as long as only the camera and not the objects themselves move in the scene. This tracker follows the principle of the tracking-by-detection paradigm, which has become increasingly popular in the field of Multiple Object Tracking (MOT)[31].

Using the same principle as in the post-processing module (3.3), the compensation done on the frames is also applied on their associated detection map. The goal of the tracker is to instantiate the cracks in a received detection map and check if it can associate this crack instance with an already established track. If that is not the case, a new track needs to be established. A track contains the subset of pixels associated with a crack and the ID of the crack instance that first established it. If newly detected crack pixels are connected to a track directly or indirectly through their associated crack instance, the track is expanded to integrate those pixels. Therefore, as long as a track is entirely inside the view, it can only grow in size and not shrink over time. It first shrinks when it passes the edges of the viewing window or is entirely occluded over a longer time. If two tracks become connected to each other, they merge, and the ID of the oldest one is continued.

The associated tracker is also saved in the frame objects in the stack, alongside the frames and their associated detection map. This means that the tracks also need to be compensated the same way as the video frames are. When a new frame gets into the queue, its associated detections are instantiated. The crack instance is then associated with the track closest to it in time, given that any exists. Using a stack of tracks can provide more stability to the track. A track is kept alive up til n frames after the last time a crack pixel was associated with it. A deeper stack is, therefore, necessary when the tracker is based on a crack detector that has a lot of FN. A bigger stack also allows tracking parts of cracks outside of the frame window for a longer time. The downside with a bigger stack is that the tracker becomes more susceptible to errors in MC. When the underlying detector has a lot of FP, these will also be picked up as crack instances and potentially become parts of tracks. The tracker can be applied after MID module and post-processing module, where the detection result most likely is more reliable.

If the MC is unsuccessful and the error is appropriately detected, all the tracks are terminated. In such an event, the tracker is reset, and a tracked crack instance will have an identity switch. If such an error is not detected, the tracker will continue. However, the track will be corrupted, and the tracking results will be faulty.

Tracker implementation

The implementation of the tracker consists of three main steps. An overview of the tracking pipeline can also be seen in Figure 3.14.

1. Cluster together all connected pixels and instantiate each crack pixel cluster
2. Associate the current crack instances with the first previous track that overlaps
3. Merge the overlapping parts into the track, keep the oldest ID.

First, the tracker receives a binary detection map where white pixels are a classification. A dilation with a small kernel size 2×2 is run over the binary map. It is important to be careful when using dilation since this process generates new information that might not be accurate. This is why the kernel size is intentionally as small as possible. Dilation is done to achieve two things. Firstly it closes tiny gaps there might be between pixels. Secondly, it gives the tracking process a bit more wiggle room when associating an instance with a previous instance. The dilation process is not necessary for the tracker to work, but it gives overall smoother results.

A *Connected Component Analysis* (CCA) is done on the dilated binary map. This groups all the white pixels into crack instances based on pixel connectivity. The CCA used in the tracker is based on the BBD T algorithm, which is 8-way connectivity based [9]. One alternative would be to use an instance segmentation network. However, this is computationally more expensive and does not make much sense in the crack domain. If two crack instances connect with each other, they can be assumed to belong to the same crack instance. Cracks are stationary objects and will not move in front of each other. Two separate crack instances could still connect given specific camera angles and perspectives. However, for all intents and purposes, CCA should be sufficient for the tracking.

All the different components are firstly treated as a new track instance of the detection. The ID is given by a global counter that increments with one for each new track instance. The next step is to associate this track with a previous track, if any exists. Each frame object in the FIFO queue used in the motion compensation includes the tracks found for the given frame. Each time a new frame object is added to the queue, the previous frame objects, detections, and tracks are updated to ensure correct pixel-to-pixel association. All tracks that belong to the same track should overlap each other.

For the development of the tracker, it is assumed that the detection it receives are correct. However, the nature of this implementation makes the tracker also able to filter out FN that are not connected to a tracked crack and are only visible in a single frame.

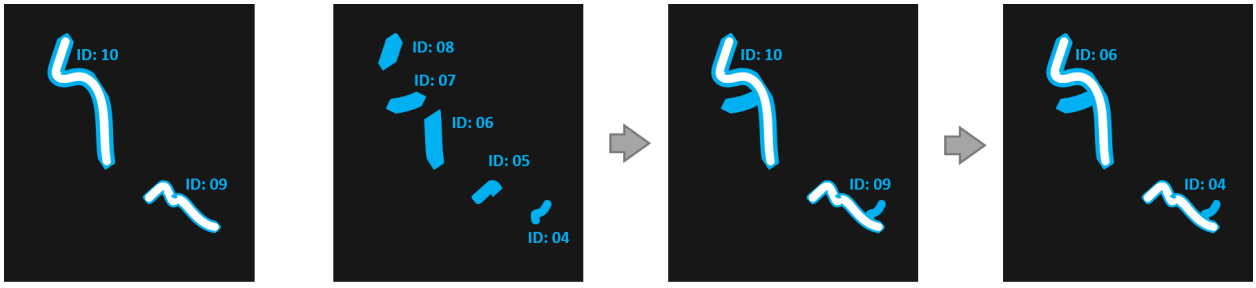


Figure 3.14.: White pixels are detected crack pixels; blue pixels the tracks/crack instances. Here it is assumed a stack depth of 2, meaning that the tracker only looks for tracks in the previous frame ($Tracks_{(i-1)}$). First, a new track is created out of the instantiated cracks ($Tracks_{(i)}$). The cracks are then associated with the tracks in the previous frames. Overlapping tracks are merged. The ID of the merged tracks becomes the ID of the oldest track that is merged in.

4. Experiments and Results

This chapter contains the experimental plan and the results. The experiments are conduced on a NVIDIA Tesla v100 graphics card with 16GB memory.

4.1. Experimental plan

All the images and corresponding label maps in the used dataset have a resolution of $342px \times 512px$. This is mainly because all the images in the publicly available crack datasets have that resolution. The data in the DNV dataset, on the other hand, initially have varying dimensions. Theoretically, The Fully Convolutional Network (FCN) architectures enables models to train on images with different resolutions. However, when training in batches all the images in a batch need to have the same dimensions. Therefore, all the DNV data is scaled (bilinear interpolation up scaling/down scaling) to fit the specified dimensions.

A total of 600 images are taken out from the dataset and put into a test set for later use. The rest of the dataset is split into a training set and a validation set, with a ratio of .8 and .2, respectively. The validation set is used to fine-tune parameters during training. It can also give an intermediate status on how well a model performs. Both the DNV dataset and the publicly available datasets are represented. Training is done in batch sizes of 8 and runs over multiple epochs until the validation loss shows the model has converged.

Originally the plan was to use the 600 images in the test set to generate synthetic videos for metric evaluation. However, in order to evaluate temporally stability, it is essential to do a visual evaluation as well. Because of time constraints, the test set is reduced to only 50 images used to generate 50 synthetic test videos. Each of those videos have a max length of 50 frames. More precisely, the synthetic test videos are generated on the fly, like in the training of the MID module models (3.4.1). Also, emulated motion blur and video compression artifacts are added randomly to the frames. Thus, the whole synthetic video generation is pseudo-random with a fixed seed. The resulting synthetic videos are, therefore, the same in between tests.

4.1.1. Available real videos

A total of five videos of inside ship tanks are made available for this project by DNV. These videos are referred to as videos A, B, C, D, and E. There are a lot of 3D structures in the scenes in the videos A, B, and E. A and B are filmed with a drone camera, while C, D, and E, are filmed with a handheld camera. Video C and D contain a lot of planar surfaces. They also are the ones with the most apparent cracks. An overview of the video properties is given in Table 4.1.

Video	Nr. Frames	Original Resolution	Type	3D scene	DNV detection
A	1770	1920 × 1080	Drone	Yes	Yes
B	930	1920 × 1080	Drone	Yes	Yes
C	930	1920 × 1080	Handheld	No	Yes
D	570	1920 × 1080	Handheld	No	Yes
E	870	1920 × 1440	Handheld	Yes	No

Table 4.1.: Table of the available real life videos. 3D scene indicates if the scene is mostly planar or has a lot of 3D elements. DNV detection shows if the detection result from a DNV detector is available or not. All the videos have a frame rate of 30 fps.

All the videos, besides *E*, have the detection result from a DNV detector associated with them. However, none of the frames in these sequences are associated with having any ground truth data. Therefore, the main test set contains the 50 synthetic video sequences, as each frame's ground truth is needed to calculate useful metrics. However, real-life videos are still used when evaluating the MC and in the visual evaluation.

When real videos are run through the pipeline during the experiment, each frame is cropped to the dimensions of 342×512, starting from the upper left corner. This was mainly done so that the real-life video, matches the dimensions of the synthetic videos. Cropping is used over down-scaling, since scaling makes the features smaller. Additionally, a smaller dimension size cuts down on the inference time during the experiments. However, the FCN architecture could theoretically handle the original frame dimensions of the videos as input.

4.1.2. Plan for testing the motion compensation

Since many of the modules depend on the Motion Compensation (MC), it is essential to know how well it actually performs. *Interframe Transformation Fidelity* ITF is used to measure the performance. ITF is one of the most popular metrics used to assess the performance of video stabilization [10]. MC, as used in this project, aims to eliminate the camera's motion in between subsequent frames. Often the goal in video stabilization methods is to smooth out the camera movement. However, motion compensation is arguably a type of video stabilization.

The ITF metric is based on the PSNR value between frames (3.2). If I is a video containing N_f frames, the ITF is defined as the average PSNR:

$$ITF = \frac{1}{N_f - 1} \sum_{t=1}^{N_f - 1} PSNR(t) \tag{4.1}$$

Here $PSNR(t)$ is based on the mean-square-error (3.1) of the peak signal-to-noise ratio, which is measured in dB, between the two frames $I(t)$ and $I(t+1)$. The idea is that the better the camera movement is compensated, the more similar compensated subsequent frames should be. Suppose subsequent frames are perfectly motion-compensated throughout the video, and there are no moving objects. In that case, the ITF score should go towards infinity [1]. However, this will never be the case during testing since visual artifacts are a part of the synthetic videos.

The motion compensation developed in this project is based on the estimation of 2D transformation matrices between subsequent frames. The four types of 2D transformation matrices used are translation matrix, similarity matrix, affine matrix, and homography. Each of those transformation

[1] ITF is calculated using OpenCVs PSNR

types is tested to see if any of them perform better than the others. Additionally, the combination of all the transformation matrices is tested. This is where the transformation matrix with the highest PSNR score is used, to compensate an older frame to the most recent frame. Finally, testing is conducted on the synthetic videos and all the five videos from inside of ship tanks made available by DNV.

Plan for testing the stacked motion compensation approaches

There are two types of motion compensation approaches developed to motion compensate older frames in the stack. The first approach is the naive approach, where all the previous frames in the stack are compensated one by one to the current frame. The second implementation is when all the transformation matrices in the stack get updated whenever a new transformation matrix between the current frame and the previous frame is estimated.

Using the estimated transformation with the highest PSNR score, a test is run on both these approaches. Given a stack depth of $n = 5$, the ITF between the newest frame and the $n-1$ previous frames is calculated. The idea is to log the result for the frames at every stack position to see how well the motion compensation performs the further back in time a frame is.

4.1.3. Plan for testing the post-processing module

The post-processing module is about post-processing the detection result from an underlying detector to improve the overall detection result. This utilizes a stack of n frames, containing the $n-1$ previous frames that are motion-compensated to the current frame. The same transformations used in the compensations are also applied to the detection results, giving the post-processing a stack of compensated binary detection label maps to work with.

Post-processing is implemented in two main ways. One is a simple voting approach, while the other is based on Neural Networks (NN). A base crack detector is needed to test the performance of these two implementations. For this, the DeepLabv3 based Single Image Detector (SID), as described in 3.4.2, is used. The SID also becomes the baseline detector when evaluating the test set of synthetic videos.

The voting approach is firstly tested separately from the NN approach, with different stack depths(n) and threshold(λ) values. The four NN models trained for the post-processing module, as described in 3.3, are; SmlNN, SmlSkipNN, UNetNN, and DLv3NN. The training data, as discussed in 3.3.3, only consists of still image label maps that are augmented to emulate a crack detector behavior on a stack. Augmentations include the emulation of false positives and false negatives. Additionally, a slight offset of $\delta = 4px$ is set to emulate the inaccuracies the MC might have.

The models are evaluated the same way as the voting. Using the SID, detections are obtained from the synthetic test videos, and a compensated stack of detection maps with depth $n = 5$ is created. This stack is then used as an input to run inference on the post-processing modules NNs. The outputs of the different post-processing approaches are compared to the output of the SID. Additionally to retrieving different metrics, the different approaches are also timed to get an overview of their respective execution time.

While the metrics give an overview of the model performance regarding the ground truth, they do not directly evaluate how temporally stable a detection is over multiple frames. Therefore, a visual comparison between the SID and the different approaches is also conducted on the test set of 50 synthetic videos.

4.1.4. Plan for testing the MID module

The MID module is about improving the detection and temporally stability with a Multi-Image Detector (MID). Using a modified DeepLabv3 with ResNet101 backbone architecture, a model with a total of 15 input channels is trained. This corresponds to a stack depth of 5 where each frame has RGB channels.

During training, the still images from the crack dataset are used to generate five-frame long synthetic video sequences on the fly. The last $n-1$ frames in the stack are motion-compensated to the first frame using the 2D transformation with the highest PSNR score. If the PSNR score is under a given threshold, $\lambda = 15$, the motion compensation is considered unsuccessful. If that is the case, the current frame is copied 5 times to create a new stack. Regardless of whether the motion compensation is successful or not, the following augmentations are added randomly in-between frames as part of the synthetic video generation pipeline: motion blur, minor compression artifacts, and altering brightness/contrast/saturation.

Training is conducted until the loss starts to flatten out. To test the MID's performance, the same test set, metrics and visual evaluation, as when testing the post-processing module, are used. The confidence for a detection to be valid is set to .5.

Motion compensation is used during training so the model can focus on the differences in crack appearances rather than also considering the movement in-between frames. As discussed earlier, the MC is not expected to be accurate. This means that the model most likely will try to learn to deal with the inaccuracies of the particular motion compensation used. It is interesting to see if the model would learn to compensate motion on its own, in addition to considering pixel appearance variation. Therefore another MID is trained, this time without doing motion compensation to the input stack. The performance of this model is measured and evaluated the same way as for the other MID.

4.1.5. Evaluation metrics for the post-processing module and the MID module

When running the synthetic test video sequences through all the approaches, in the post-processing module and the MID module, the number of pixels that are: False Positive (FP), False Negative (FN), True Positive (TP), and True Negative (TN), are logged for each frame. Using these numbers, interesting metrics that can give an insight into the individual approaches behavior can be calculated:

Sensitivity, aka. recall hit rate, or True Positive Rate (TPR), measures the proportion of correctly identified positives. In this case, it means the proportion of "crack pixels" that are correctly classified.

$$TPR = \frac{TP}{TP + FN}$$

Specificity, aka. selectivity or True Negative Rate (TNR) measures the proportion of the correctly identified negatives. In this case, the proportion of "non-crack pixels" that are correctly classified.

$$TNR = \frac{TN}{TN + FP}$$

Precision, aka. Positive Predicted Value (PPV) measures how many of the predicted positives actually are positives. In this case, how many of the predicted "crack pixels" are actually crack pixels.

$$PPV = \frac{TP}{TP + FP}$$

DICE, aka F1 score, can be described as the harmonic mean of precision and sensitivity. The DICE score is great when measuring an uneven class distribution where there are many actual negatives, which is the case for the test-set used in the experiments.

$$\frac{2TP}{2TP + FP + FN}$$

*(*Intersection over Union (IoU)), aka Jaccard score, measures the similarity between the prediction and the ground truth.

$$\frac{TP}{TP + FP + FN}$$

Both the DICE and IoU scores are popular in the field of image segmentation. They are therefore used as the primary evaluation metric to see how well an approach does overall. From their respective definitions, it becomes clear that both metrics will always be within a factor of 2 of each other.

$$DICE/2 \leq IoU \leq DICE$$

It is also evident that the scores have the same extreme points, with 1 being a perfect match and 0 being a complete disjoint. The metrics are positively correlated, meaning that if detector A is better than detector B under one metric, it will also be better under the other metric. Initially, the two metrics can seem interchangeable. However, the difference becomes evident when quantifying how much worse a detector A is over a detector B when calculating the average score on a set of inferences. The IoU metric usually penalizes FP and FN more than the DICE metric. This makes the DICE metric measuring more something of an average performance, while the IoU measurement is closer to a worst-case performance. By using both metrics, the performance evaluation can be more insightful. Note that there are extreme cases where the errors of both metrics can become misleading. For example, if the ground truth label only contains one pixel and one pixel is detected, but one FP pixel is detected, the DICE score would be $2/3$, while the IoU score would be $1/2$. These errors can significantly influence the average score calculated over the test set, which is why it is so important to log the FP, FN, TP, and TN at each frame.

4.1.6. Plan for testing the tracking

The implemented tracker is visually evaluated on the 50 synthetic test videos. While the tracker mainly uses the output from the voting approach with a stack depth of $n = 5$ and $\lambda = 3$, it is also evaluated using the real videos, paired with DNV's detector. During the visual evaluation, it is essential to look out for identity switches of a track and the inaccuracy of a track over time.

4.2. Results

This section presents the main results of the experiments. Further results can be seen in Appendix B.

4.2.1. Results for motion compensation

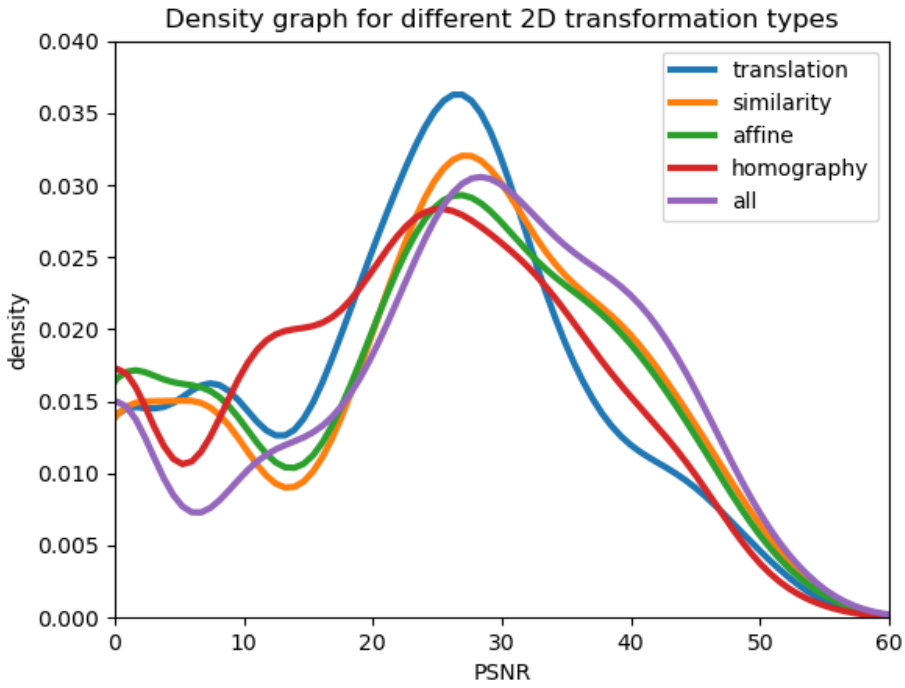


Figure 4.1.: Density graph the PSNR score for the frames in the 50 synthetic test videos, when using different types of 2D transformation. The different colors represent the different transformation approaches

Transformation type	ITF score (dB)
translation	23.19
similarity	25.27
affine	24.15
homography	22.68
all	26.42

Table 4.2.: The average ITF score for all the frames in all the tested videos (synthetic and real)

Result for Naive vs Update approach

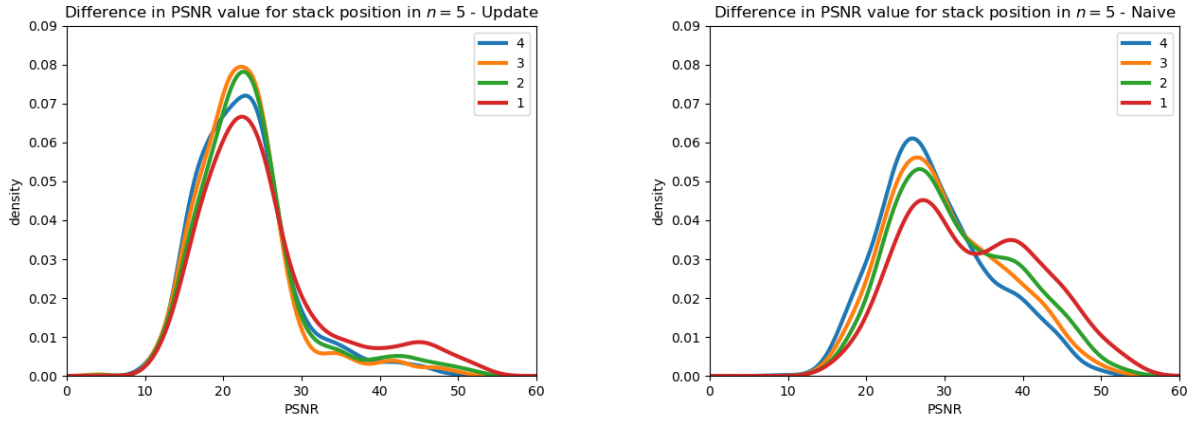


Figure 4.2.: Density graph for update and naive motion compensation approach based on the PSNR values for all the frames in the synthetic videos, given that the stack actually had a depth $n = 5$. The legends 1 to 4 indicate the frame position in the stack, where 1 is the closest and 4 the frames the furthest away in time to the current frame.[2]

Frame position	ITF - Update	ITF – Naïve
1	25.06	33.58
2	24.71	31.94
3	23.87	30.71
4	22.88	25.83

Table 4.3.: The ITF values (average PSNR) values for the two motion compensation approaches, run on the synthetic videos, measured in dB. The motion compensation chooses the best transformation at each frame. The higher the value the better the compensation

Results for different 2D transformation types

The following experiment results are the average results of all videos (real-life videos and the synthetic test videos).

[2]The naive approach has a full stack in 44% of all the frames while the update approach has a full stack in 60% of the frames

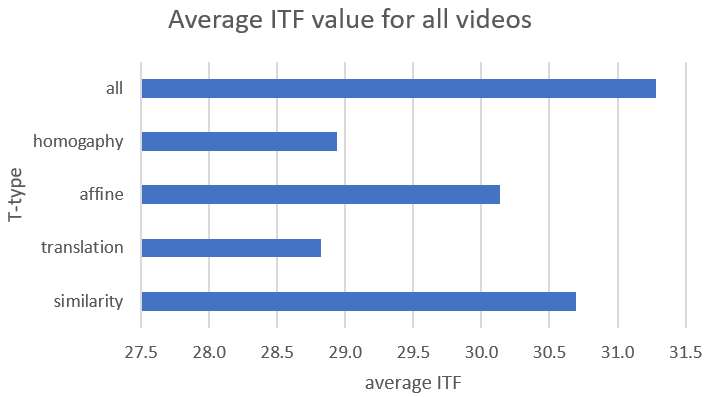


Figure 4.3.: The average ITF value (based on compensation of the previous frame to the most recent frame) for the different transformation types. Further break down of this chart can be seen in B.8.

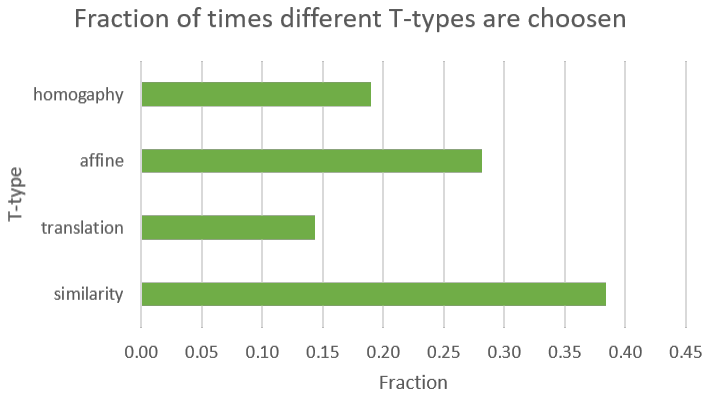


Figure 4.4.: Fraction of times different 2D transformation types are chosen, when always choosing the best 2D transformation at each frame. Further break down of this chart can be seen in B.10.

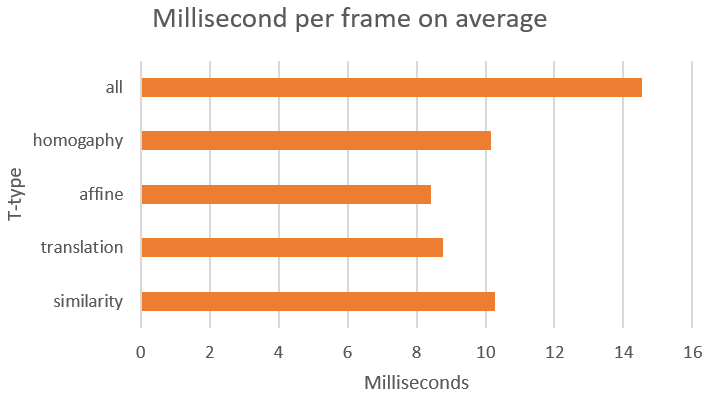


Figure 4.5.: Average time to to calculate the transformation types between the most recent frame and the previous frame.

4.2.2. Results for different voting parameters

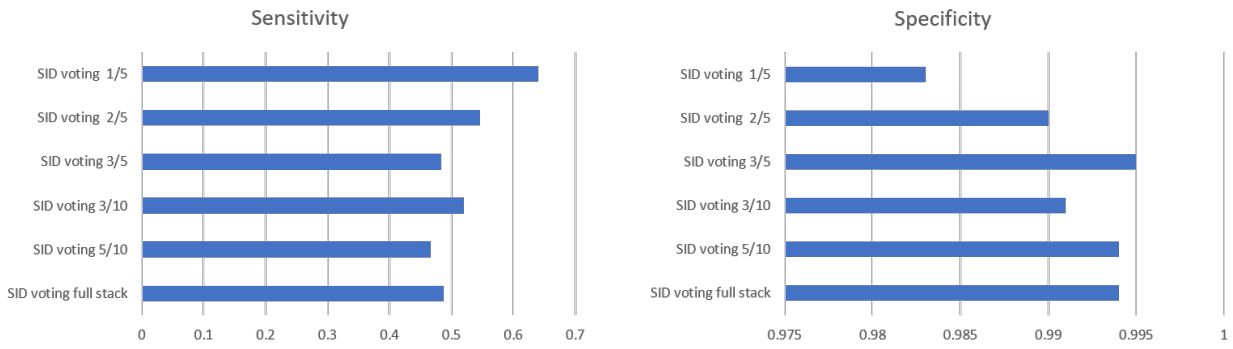


Figure 4.6.: Sensitivty and Spesificity of different voting approaches

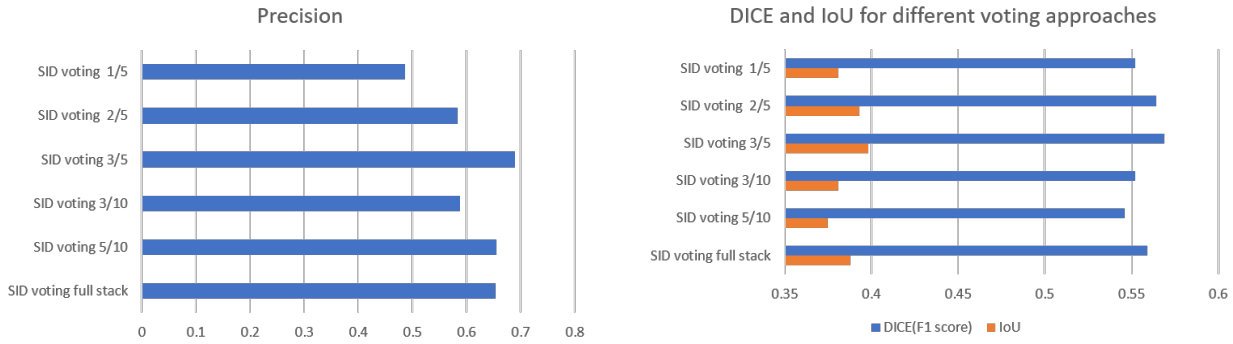


Figure 4.7.: Precision DICE and IoU of different voting approaches

4.2.3. Results for post-processing module and MID module

SID		Predicted	
		Positive	Negative
Actual	Positive	0.014	0.011
	Negative	0.004	0.972

(a) Single Image Detector (SID)

SID voting		Predicted	
		Positive	Negative
Actual	Positive	0.012	0.013
	Negative	0.005	0.970

(b) Post-processing SID with voting, where $k = 5$, $\lambda = 3$

SID small net		Predicted	
		Positive	Negative
Actual	Positive	0.019	0.006
	Negative	0.038	0.936

(c) Post-processing SID with small network

SID skip net		Predicted	
		Positive	Negative
Actual	Positive	0.016	0.008
	Negative	0.017	0.959

(d) Post-processing SID with small network with skip connections

SID UNet		Predicted	
		Positive	Negative
Actual	Positive	0.014	0.010
	Negative	0.009	0.967

(e) Post-processing SID UNet based network

SID DLv3		Predicted	
		Positive	Negative
Actual	Positive	0.014	0.011
	Negative	0.007	0.969

(f) Post-processing SID with DeepLabv3 based network

MID w/ MC		Predicted	
		Positive	Negative
Actual	Positive	0.013	0.012
	Negative	0.004	0.972

(g) Multi Image Detector (MID) with MC

MID no MC		Predicted	
		Positive	Negative
Actual	Positive	0.013	0.011
	Negative	0.003	0.973

(h) MID without MC

Figure 4.8.: Confusion matrices for the different approaches

	Sensitivity	Specificity	Precision	DICE	IoU
SID	0.558	0.996	0.789	0.654	0.485
SID voting	0.484	0.995	0.689	0.569	0.398
SID small net	0.765	0.960	0.321	0.452	0.292
SID skip net	0.678	0.983	0.499	0.575	0.403
SID UNet	0.572	0.991	0.606	0.588	0.417
SID DLv3	0.558	0.993	0.658	0.604	0.433
MID w/ MC	0.522	0.996	0.772	0.623	0.452
MID no MC	0.536	0.997	0.828	0.651	0.482

Table 4.4.: Table of sensitivity, specificity, precision, DICE and IoU for each of the models. The yellow fields indicate the best score in each column. Calculation is based on the confusion matrices of the number of pixels, see appendix C.1

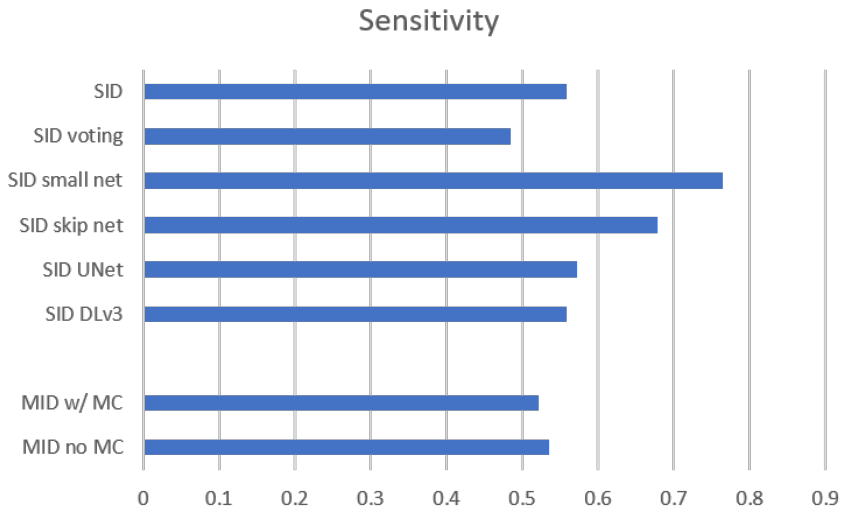


Figure 4.9.: Chart for comparing the sensitivity of the models, based on Table 4.4

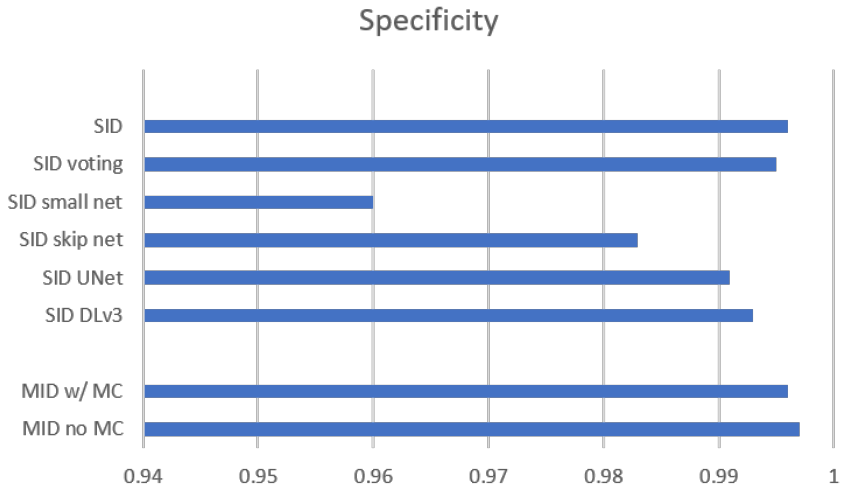


Figure 4.10.: Chart for comparing the specificity of the models, based on Table 4.4

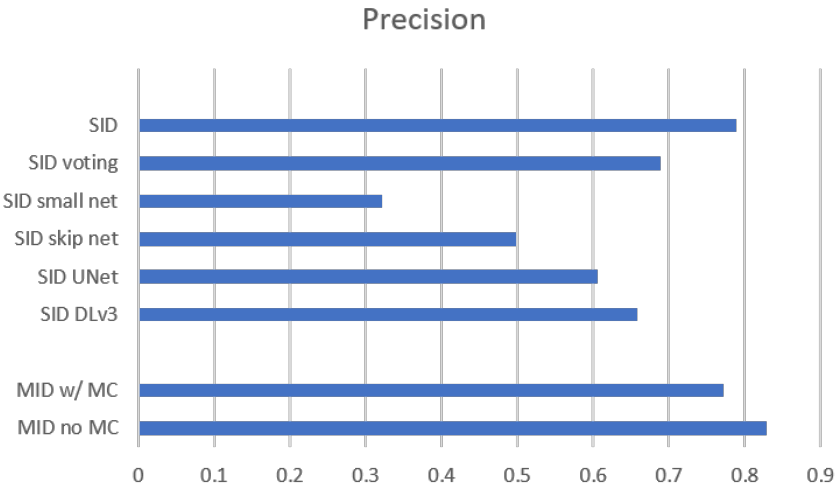


Figure 4.11.: Chart for comparing the precision of the models, based on 4.4

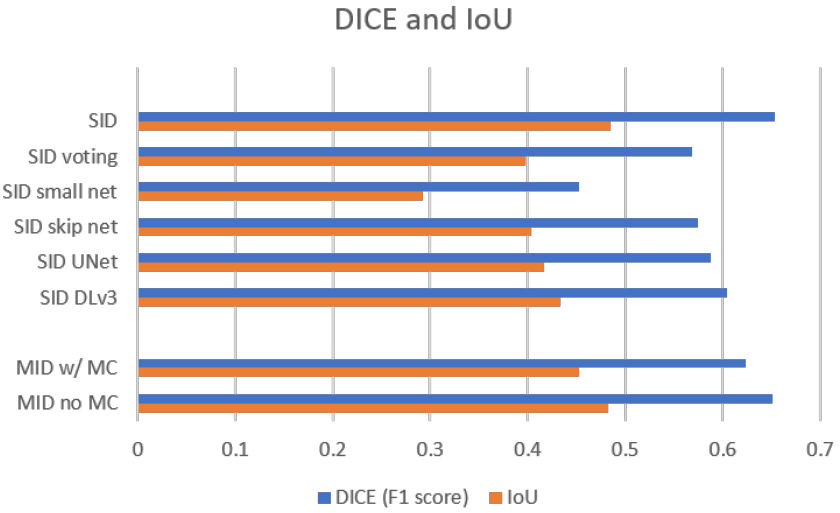


Figure 4.12.: Chart for comparing the DICE and IoU of the models, based on Table 4.4

4.3. Interesting observations in the visual evaluation

This section displays some of the interesting observations found during the visual evaluation of the different models. If nothing else is indicated, the general color coding is as follows:

1. Blue color: FP outputs compared to GT
2. Red color: FN outputs compared to GT
3. White color: TP outputs compared to GT

Note also that the frames are darkened to give a higher contrast to the color-coding.

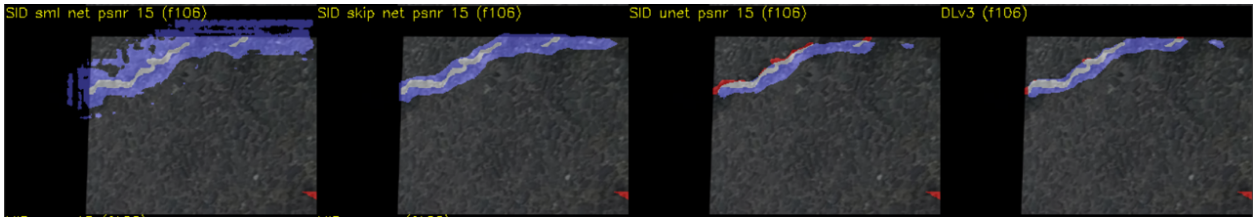


Figure 4.13.: Comparing outputs for the 4 post-processing NNs at frame 106 in the synthetic video.

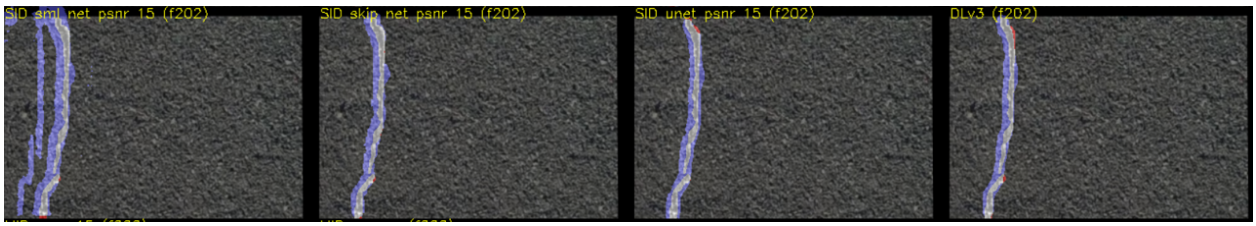


Figure 4.14.: Comparing outputs for the 4 post-processing NNs at frame 106 in the synthetic video.

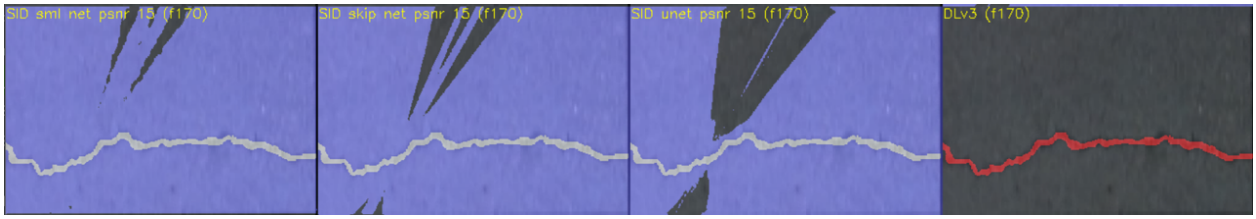


Figure 4.15.: Comparing outputs for the 4 post-processing NNs at frame 170 in the synthetic video.

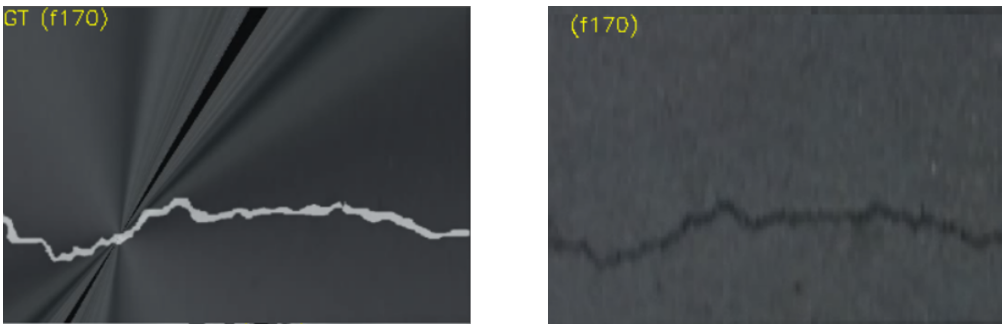


Figure 4.16.: On the left: Overlay of the previous frame of 170 in the synthetic video, that is "motion compensated", to fit frame 170. This bad motion compensation is a rare case where the PSNR value is way higher than it should be. The white line indicates the GT
On the right: The frame 170: Note the lack of detail and similar colors to the other Figure

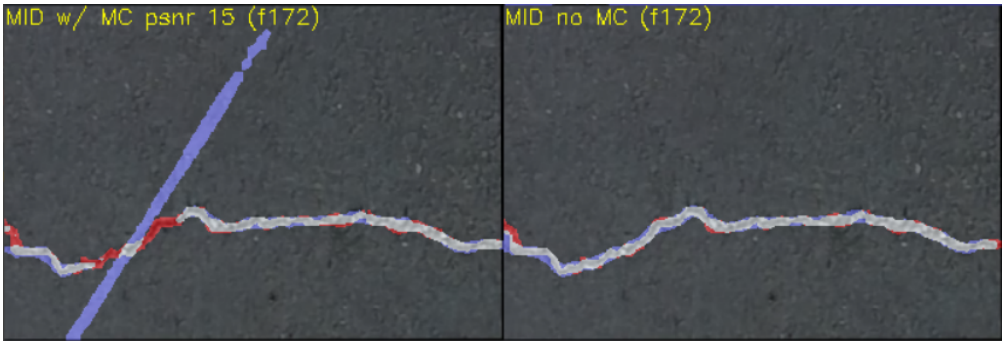


Figure 4.17.: The undetected faulty MC having its impact on the MID-with-MC, but not the MID-no-MC

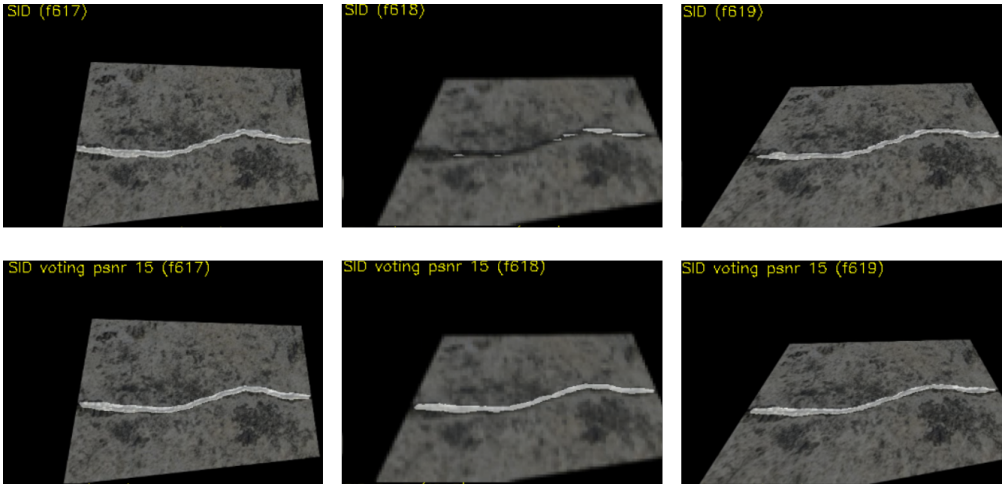


Figure 4.18.: The output from the SID (top) versus the output of the SID with voting (bottom) from frame 617 to 619 in the collection synthetic test videos. Note how the voting smooths out the detection result in frame 618. Here the white pixels are the output

4.3.1. Tracking results

Figure 4.19.: The output of the tracking module on video "C" at frame 95, 96 and 150. The crack instance ID 10799, stays consistent

5. Discussion

This chapter contains the overall discussion of the metric results and visual observations made in the experiments.

5.1. Motion Compensation Discussion

The motion compensation component (MC) is a crucial part of this project and serves in many ways as the backbone for all the modules. Visually the MC seems to work fine in most scenarios where the frames are detailed, and the scenes are planar. This is expected as the motion compensation developed here is based on the estimation of 2D transformation matrices. As the MC uses descriptor-based matching, there must exist enough unique details in-between frames for such estimation to work correctly. While the MC overall seems to perform well, some scenarios make it struggle. For instance, a lot of camera movement can lead to less spatial information from a previous frame being represented in the current frame. This gives the MC fewer overlapping features to extract to use for transformation matrix estimation. The less points there are in the estimation, the more often the results become inaccurate. Rapid movement often leads to motion blur and video compression artifacts. This, in turn, also makes it harder to match spatial information between two frames since the appearance of the spatial information can have been altered drastically.

5.1.1. Best 2D transformation type

The average Interframe Transformation Fidelity (ITF) value for all the tested videos, shown in Figure 4.3, suggests that choosing the best available transformation type at each frame ("all") gives the best motion compensation results. By its definition, this outcome is expected and simply confirms that this approach works. More interesting is to see the individual performance of the different transformation types. In general, the differences are not significant. Translation, the transformation type with the fewest parameters with only 2 Degrees of Freedom (DoF), performs the worst. This is unsurprising since motion compensating by only translating does not account for rotations and perspective changes. More unexpected is that its performance is similar to the homography transformation. Homography has the most DoF with 8. In theory, this should make motion compensating a previous frame to the current frame more accurate. However, with high DoF, there comes a high amount of parameters. Estimating many parameters leaves room for more errors and minor inaccuracies. Affine and similarity transformation, with 4 DoF and 6 DoF, respectively, have the best performance. Both are offering a compromise between the number of parameters to estimate and DoF.

The performance of the individual transformation types is further confirmed when breaking down the "all" type, shown in Figure 4.4. It can be seen that the similarity transformation is chosen almost 40% of the time, while the translation transformation is only chosen around 15%. Figure 4.5 shows the average execution time per frame of the different transformation types, measured in millisecond. Keep in mind that time measurements are depend on the hardware the program is running on. Nevertheless, it becomes clear that while the individual transformation types have a similar execution time, the combination of them gives a small time penalty during motion compensation.

All the other transformation types are tried out during the motion compensation with the "all" transformation, and their PSNR value is compared. Since the time difference between the "all" and the others is small, it becomes evident that this process does not have much overhead. The small time difference also suggests that most of the average execution time of transformation calculation is not used for the calculation itself but rather for fetching the frames.

Videos used in the experiments are predominantly planar. More video footage from actual drones in the target environment is needed to see which transformation type is better. As stated, the ensemble method ("all") does not have much overhead. Additionally, it guarantees to consistently produce the best MC, given that the self-evaluation works correctly.[1]

5.1.2. Difference between naive and update approach

The motion compensation should work between the previous frame and the current frame and for frames further back in time. This project investigated two methods to motion compensate older frames: the update and the naive approach. The update approach propagates the transformation between the previous and the current frame, down to the older frames. This makes it overall more robust to extreme movement. The naive approach relies on the assumption that enough spatial information in the older frames also exists in the current frame to estimate a robust transformation matrix. If the camera moves fast, this assumption will not always hold up. Suppose there is slight camera movement in-between frames. In that case, the naive approach is more robust since inaccuracies in the transformation estimation between the previous and the current frame do not propagate down. However, when there is a lot of movement, the update approach is slightly better since it does not rely on a minimum amount of spatial information to overlap between older frames and the current frame.

During development, the update approach was first implemented, and initially, there were no plans to try the naive approach. This is why the update approach has been used throughout this project. When comparing the ITF's for the two approaches (Table 4.3), the naive approach overall performs better. The ITF needs to be measured for all the previous frames in a stack of size $n = 5$. Therefore for a fair comparison between the different positions in the stack, the PSNR value of the frames are only measured when the stack is full and successfully motion-compensated. Note that 60 % of the frames were associated with a full motion-compensated stack in the update approach, while there were only 44% in the naive approach. This means that the results for the update approach include the PSNR value of more frames. It can be assumed that many of these extra frames the update approach evaluated are harder to motion-compensate, leading to lower ITF numbers. Therefore, it is important not to compare the numbers between the update and the naive approach directly but rather focusing on the behavior of the different stack positions. The way the naive approach is implemented, the stack resets as soon as one of the 4 previous frames fails to motion-compensate to the current frame. This is the reason the stack resets more often in the naive approach. It can be considered only to remove the parts of the stack that had a failed motion compensation in future implementations.

Density graph difference between naive and update approach

Looking at the density graph for the update and the naive approach in Figure 4.2, it becomes clear that the older a frame is, the slightly worse the motion compensation usually is. A density graph is a smoothed histogram that shows the distribution of a numeric value, in this case, the PSNR.

[1] Some of the real-life videos do future a lot of 3D structures. Additionally to no being planar the assumption that all objects are stationary is also not met (see B.3)

Therefore the area under the curve is equal to 1. The more of the area under the curve is to the right in the plot, the better the motion compensation is on average. The red graph, which represents the previous frame, is furthest to the right, while the blue graph representing the oldest frame in the stack is the furthest to the left, in both. This phenomenon that the newer a frame is, the better the motion compensation is to the most recent frame can also be confirmed when looking at the ITF values in Table 4.3. The difference between older frames and newer frames seems to be more prominent in the naive approach. This can mostly be traced back to rapid camera movement. If the camera moves fast, the older frames might not have much spatial overlap with the current frame anymore, making the compensation for those frames harder. The motion compensation itself only operates on the previous frame and the current frame in the update approach. Since the other frames are updated based on this outcome, it is not important for older frames to have overlapping spatial information with the current frame. In theory, older frames can even move out of the viewing window and still be motion-compensated to the current frame. This makes the update approach overall more robust to large camera movements. However, it is worth noting that all the motion compensation errors between the previous frame and the current frame propagate down in the stack. In the naive approach, the motion compensation is done between all the older frames in the stack and the current frame, preventing error propagation. Even if one frame in the stack is motion-compensated incorrectly, all the other frames can still be correct. This makes the naive approach in theory more robust to errors in the motion compensation process, given that there is less movement. However, the naive approach also comes with more overhead, since updating previous transformation matrices is computational less expensive than recalculating a transformation matrix. There is also the option to do an ensemble method of both approaches, choosing the output with the highest PSNR score for each frame in the stack.However, this is yet to be implemented.

5.1.3. Self evaluation of the motion compensation

When motion compensation is not deemed successful, all the feature operations relying on the motion compensation are terminated. During the visual evaluation, it becomes clear that while the motion compensation generally seems to evaluate itself well, a few cases of bad motion compensation are undetected. This has drastic consequences for the performance of the different approaches, as can be seen in Figure 4.15. In this example, the compensation fails completely. The first three post-processing NNs, are not able to handle such a corrupted stack at all. The DLv3NN, however, simply ignores the stack and outputs nothing (The red line shows FN, meaning the SID-DLv3 outputs nothing). However, the DLv3NN most likely only handles it in this situation since it gets contradicting inputs. If the inputs are not contradicting each other, the DLv3 has no way of knowing that it receives corrupt information.

Undetected motion compensation failure

Those corrupted motion-compensated stacks are rare, but they can happen. Extreme errors in the motion compensation that slip through undetected, like shown in Figure 4.16, were primarily observed in the synthetic videos. However, it is also observed in real-life scenes seemingly often caused by rapid movement and motion blur. How well the motion compensation does, is as described in 3.2, measured with the PSNR value between two subsequent frames. The higher the PSNR value, the more similar the two subsequent frames are to each other. On the other hand, when the PSNR value is low, the motion compensation is most likely bad. Note that also motion blur and other changes in-between frames can contribute to a low PSNR score. For the motion compensation in this experiment to be considered successful, the PSNR score must be at least 15dB. This threshold was set during development as it seemed overall to give consistent feedback if a motion compensation

was bad or not. If there are no similarities between two subsequent frames, the PSNR score goes towards 0dB. So how did the PSNR score in Figure 4.16 get over that threshold?

The most recent frame has some motion blur in it and lacks a lot of detail. This already indicates that the motion compensation might struggle since finding enough spatial points and descriptors that are unique (points that cannot be confused with each other) becomes hard. The motion compensation does indeed fail, but the PSNR value is still over the threshold. Visually the wrongly motion compensated previous frame as seen in left in the Figure 4.16 and the current frame, seen on the right, do not look similar. That is to say that their structure and spatial appearance do not look similar. The grayscale values, on the other hand, do. The lack of detail and similar grayscale intensity makes the PSNR value higher than it should be. The PSNR value is calculated on grayscale frames in this project. While it is slightly less computationally expensive, it might be more susceptible to such errors. In this particular situation, the calculated PSNR value alone is not sufficient to determine if the motion compensation was successful or not. This also explains why this kind of false acceptance of the motion compensation are less observed in the real-life videos. Often there are enough details and different structures in each frame in these videos so that the spatial feature extraction works well. Additionally, those variations in spatial information can lead to that a failed motion compensation does not get such a high PSNR score.

PSNR value threshold for self evaluation

The self-evaluation has been observed to generally work well on real-life videos. Sometimes inaccuracies in the motion compensation, occur but these are primarily small. These inaccuracies can often be linked to the fact that 2D transformations are estimated for a 3D scene. Also, a lack of detail and video artifacts, which often result from rapid motion, can throw the compensation and/or its evaluation off. How sensitive the motion compensation is, is set by a PSNR value threshold.

In the experiments the PSNR value threshold is set to 15dB. However, the higher the threshold is set, the more self-critical the motion compensation is. What the most optimal PSNR threshold is may depend on the scene and the video. If there is a lot of motion blur, changing lights and shadows, and compression artifacts in a video, even when the motion compensation theoretical could be perfect, the PSNR value would still be low. This is because the PSNR value is appearance-based.

5.2. Evaluation of the post-processing module

From the numeric results in Table 4.4 it seems like non of the approaches developed in this project actually outperforms the Single Image Detector (SID). It is important to remember that the metrics are only based on averages of all the frames in the synthetic video. As discussed in 5.1.3, the synthetic videos are more prone to issues in the motion compensation. When visually evaluating the different approaches, it becomes clear that they do indeed offer some temporally detection stability in subsequent frames to the output of the SID (i.e., see Figure 4.18). Therefore it is crucial to see the numeric results in the context of the visual evaluation, which was conducted on all of the approaches.

Post-processing by voting mostly smooths out the detection results across frames. An example of the voting approach doing this can be seen in Figure 4.18. This also holds for all the NN approaches in the post-processing module, besides the SID-sml. However, the results can only be smoothed as long as a motion-compensated stack exists. If the stack is reset often due to motion compensation failure, the post-processing output is the same as its input.

Looking at the DICE and IoU metric in table 4.4, voting has a similar performance to the other

networks, besides the poor performing SID-sml. Voting has the lowest sensitivity of all the approaches. There are several reasons why this makes sense. It has been disclosed earlier that the motion compensation is not expected to be perfectly accurate. The voting approach only works with detected pixels in the stack that are perfectly overlaying each other. If the detected pixels are not perfectly overlaying each other, they are simply ignored. Minor acceptable errors in the motion compensation give the output of the voting approach a kind of "erosion" effect the higher λ is, making the detected area smaller than in the input. Additionally, with the delay in the detections resulting from the threshold λ, the number of positively detected pixels is often lower than in the input. With a low λ value, the opposite can be observed, where the errors caused by the MC make the detection of a crack "wider" (ghosting effect).

None of the approaches in the post-processing were fine-tuned to work with the specific SID. The SID used in the experiment turned out to be conservative, meaning its results contain more FNs rather than FPs. This is one of the main reasons why all the post-processing approaches, metric-wise, do worse than the SID (4.4). The post-processing used here requires a stack of n compensated detections. However, if there are not enough detections in the stack for the post-processing to work, the detected pixels tend to be filtered out rather than recovered. Additionally, inaccuracies in the motion compensation can throw the post-processing off.

5.2.1. Post-processing module, voting behavior

The different voting approaches, overall, seem to have similar performance when it comes to the DICE and IoU score (Figure 4.7). However, in the visual evaluation, it becomes apparent that the bigger the threshold λ, the more stable the detection is in-between frames. The downside is that there is a longer delay with a higher λ before a detection is assumed to be a positive.

Stack depth for voting

It can also be observed that a bigger stack can recover information further back in time. The problem is that the further back in the stack a frame is, the more inaccurate the motion compensation often becomes (4.2). Voting on inaccurate motion-compensated frames leads to inaccurate results. What the ideal stack depth for the voting approach is depends on how good the motion compensation is, which often depends on how much movement and video artifacts there are in the input video stream.

Voting threshold

A higher λ can, the same way it filters out FP, also filter out some of the errors generated by an inaccurate MC. Again this is at the expense of a detection delay. Even then, the voting is potentially faulty since only errors that overlap fewer times than λ are filtered out. If the temporally instability of the detection result is the consequence of flickering FP detections, a higher voting threshold λ can help to smooth this out. The flickering FP detections is a phenomenon that can be observed in the output of some detectors. Here the detector produces a FP detection in only a few frames before it disappears again. These FPs are filtered out by requiring a detected pixel to be detected in λ frames before it is accepted as a positive detection. Another downside of this approach is that it comes with a detection delay. If a crack moves into the view, the crack pixels are not registered as positive detections before at least λ frames have been sent through the post-processing pipeline. If the underlying detector is more conservative and produces more FN, the threshold λ can be reduced. Setting $\lambda = 1$ makes the overall detection extremely sensitive (see "SID voting 1/5" in the sensitivity chart in Figure 4.6). All the detected pixels in the motion-compensated stack n are

assumed to be TP. This includes detections that are only visible in one frame and normally would be labeled as FP. Therefore this kind of post-processing is unable to filter out flickery FPs. Any possible errors caused by the motion compensation are also just forwarded through. Assuming the motion compensation would be 100% accurate and a detector would never produce any FP and only FN, this voting approach with a $\lambda = 1$ would perform the best. Unfortunately, that is not the case. Therefore, this approach generally has a lower specificity and precision than the others.

When there are no frames in the stack, this can be handled in a couple of ways. One way is to let the threshold λ be dynamic and change with the number of frames in the stack. Another way is to let λ be static and require a minimum number of frames in the stack before the post-processing pipeline starts to work. The approaches "SID voting 3/5" to the "SID voting full-stack" have the same parameters $\lambda = 3$ and $n = 5$, but the latter one only does voting when the stack is completely full. This means that as long as the stack is not filled up, the "SID voting full-stack" output will be the same as the original detection. Looking at the metrics in Figure 4.6 and 4.7 the "SID voting 3/5" approach performs only slightly better. In the video evaluation, especially in the real-life videos where the MC is generally more stable, there is almost no difference between the two approaches of handling a stack. In the latter approach, the only difference is a delay before the smoothing effect occurs after a stack rest. The metrics are slightly different here because, in the synthetic videos, stack resets are more common.

Voting with inaccuracies in the motion compensation

While the voting approach in the visual evaluation seems to give overall smoother and more temporally stable detection results than the SID, the metrics such as the DICE and the IoU suggest that it actually performs worse than the underlying detector. It is important to note that these metrics originate from the synthetic videos, as these were the only videos with ground truth data available. As discussed earlier, the motion compensation seems to struggle in some of the synthetic videos, especially if there is a lack of interesting spatial information in a frame. Sometimes there are even errors in the motion compensation without the self evaluation noticing it. These errors can also clearly be seen when visually evaluating the post-processing of the detections in the synthetic videos. The undetected errors are the main reason why the metrics do not line up with the general observation. Those extreme undetected motion compensation errors exist but are not as common in real-life videos. Minor inaccuracies are expected in the motion compensation, especially when there is a lack of details in frames. Inaccuracies can also be traced back to the fact that the MC builds on the estimation of 2D transformations while operating on 3D scenes. The voting can operate despite minor inaccuracies. However, this will lead to small, often negligible innacuracies in the voted detection results. The voting is not spatially aware, and therefore if some pixels from earlier frames are not lined up perfectly with the current frame, the voting will be off (see ghosting example 3.6). The resulting errors are often observed at the edges of crack instances. Only extreme MC errors must be detected, and the voting must be terminated to prevent extreme detection errors. The voting approach $\lambda = 3$, $n = 5$ was used when comparing the other post-processing module approaches as it had a slightly better performance on average than the others.

5.2.2. Behaviour of the Post-processing Neural Networks

This spatial awareness has been one of the reasons to propose NN for post-processing. The idea is that NNs can potentially compensate for the minor expected errors in the motion compensation. There is also the possibility for NN to adapt to the errors of a specific underlying detector. While this is not tested in this project, it definitely would give overall better results. The training data used for the NNs are synthetically generated and therefore not based on the output of a specific

detector.

Looking at the four neural network-based post-processing approaches, (run on synthetic videos based on the output of SID) it becomes evident that the deeper the network, the better the result. Both the DICE and IoU metric confirm this (4.4). When visually evaluating the different outputs, this becomes even more clear. In the Figures 4.13 and 4.14, it can be observed that the deeper networks seem to be closer to the ground truth (less blue pixels). Note that all these networks motion compensate the stack of 4-previous frames to the current frame and use the compensated stack of detection from the SID as their input. This means that they heavily depend on how good the SID is and how well the MC does. They can, in theory, beat the SID if there are some instabilities in the SID detection in-between frames.

The smaller post-processing Neural Networks

The SID-sml network generally seems to generate a lot of FP artifacts, which can be seen in both Figure 4.13 and Figure 4.14. This has most likely something to do with the augmentations done on the training data for the post-processing neural networks. To emulate inaccuracies in the motion compensation, the individual label maps in a stack are randomly shifted a few pixels during training. This means that the post-processing NNs, need to learn to both compensate for this minor inconsistent spatial information and which detected pixels to keep. The SID-sml network seems not to be able to compensate for this minor shift. This makes sense, as the network's encoder-decoder architecture loses a lot of the spatial information further down in the network. This is also why the SID-skip, which has the same amount of layers, does much better. The skip connections better retains the spatial information in the decoder part of the network. This lack of spatial awareness in the SID-sml is especially evident in Figure 4.14. The blue line on the right (which indicates a FP generated by the SID-sml) has a shape that seems similar to the Ground Truth (GT). Here the SID, which the SID-sml post-processes, had a detection that was close to the GT. However, even when the motion compensation works (which seems to be the case based on the other results on that frame), the lack of spatial awareness shifts the detection result, so it no longer overlaps. The SID-sml has not learned to compensate for the minor shifts adequately.

The deeper post-processing Neural Networks

Generally, the bigger the network, the better it seems to learn how to compensate for the minor errors of the MC. The deeper networks also seem to be more conservative when it comes to accepting a detection. This is, among others indicated by that the blue pixels seem to get "tighter" around to the ground truth with increasing network depth. Overall the smaller networks have a more "dilated" appearance, increasing the detection area. This leads to an "overestimation" in their prediction, thereby capturing more True positives. This explains why the sensitivity of the smaller networks is higher (4.9), and the specificity lower (4.10). To recap, sensitivity is the proportion of correctly identified positives, while specificity is the proportion of correctly identified negatives. With the precision measurement (4.11), this trend becomes even more apparent. The simpler the network, the lower the precision score, meaning that many predicted positives are not necessarily positives.

Observations in the post-processing Neural Networks result

The smallest NN seems to lack a lot of spatial awareness, which resulted in it moving the detections seemingly randomly around in its result. Strangely enough, this effect is only observed when the SID-sml operates on the output from the SID developed in this project. If it operates on the output

from DNVs detector on real-life videos, those artifacts disappear. However, those artifacts are never seen on the other three NNs, and they all have a better DICE and IoU score than the voting approach.

Visually the NNs achieve a similar temporally stable result as the voting approach. On the synthetic videos, it is observed that the detection output from the post-processing NNs often go a little bit over the edge of a crack instance. This is the opposite effect than is observed with the voting approach when $\lambda = 3$. With deeper networks, this effect is less prominent, as they tend to be more conservative. In real-life videos, the NNs outputs do not seem to have the same dilation effect. However, it is still observed that the deepest network, the DLv3 based NN, still is extremely conservative. In fact, it is so conservative that it produces more gaps in between the crack instances than all the other approaches. The networks deploying skip-connections (SID-skip and SID-UNet) seem to perform the best when it comes to outputting temporally stable detections when used with DNVs detector. When used with the in this project developed SID, the DLv3 based architecture seemed to be the most stable.

Overestimation is generally more observed across all the networks when they use the SID over the DNV detector as their base detector. Generally, the NNs seem to behave differently, depending on which base detector they use. There is an augmentation during training of the models that emulates minor inaccuracy of a motion compensation. It was hypothesized that NNs could potentially compensate for those minor inaccuracies in the MC. This requires them to learn some spatial awareness when they operate on binary detection maps. However, it seems that overestimating slightly over the edges of a detected crack is the primary way the NNs try to combat the MC inaccuracies. While the deeper NNs seem to overestimate less, this is not necessarily linked to spatial awareness but can also simply be a result from being more conservative.

Neural Networks vs. voting

None of these networks were learned to compensate for the errors of a specific detector. This is why there is such a behavior difference to be observed depending on which base detector is used. The stack size was only set to $n = 5$, which gave the networks little information to work with in the first place. The MC runs into errors more often on the synthetic videos and resets the stack. This in turn also makes the post-processing approaches reset more often. Temporal stability can only be granted if there exist enough frames in the stack.

The NNs learn at least something since they clearly do not simply superimpose their input but deploy some sort of learned voting strategy. The rules of this strategy do not seem as linear as with the voting approach. Most likely, there is some spatial awareness and a lot of general learned behavior that makes most NNs outperform the voting approach on a stack depth of $n = 5$ (on synthetic videos using SID). The bigger stack the voting approach has to operate on the more temporally stable results it gets. It is natural to assume that this also is the case for the NNs, though this is not tested. NNs are less flexible when it comes to stack size as they need to be retrained to support a different number of input channels.

The deeper NNs contain more parameters and use more memory. However, inference wise PyTorch does a great job at optimizing, and there are only minor differences in favor of the smaller networks when it comes to execution time (see Figure B.11 in appendix for execution time table).

Both the voting approach and the NN-based post-processing give overall more temporally stable outputs. However, they rely heavily on motion compensation. Post-processing should not get a worse result than the detector it is based on, but here it clearly does. Almost all of the errors can be traced back to a suboptimal MC. If the motion compensation has errors that remain undetected,

both post-processing approaches struggle. In the voting, the stack depth and voting threshold λ parameters need to be fine-tuned to perform the best for a given detector. The NN, on the other hand, needs to be trained on a given stack depth. This means that some detection results need to be duplicated if the stack has not reached the desired depth. However, the benefit of the NNs is that they can adapt to the specific errors in the detection result of the underlying detector. If a detector is either extremely conservative or has many FP detections, the NNs can learn to handle that without the need for manually fine-tuning. The results show that which of these synthetic trained NNs performs the best depends on the underlying detector. The deeper networks do not necessarily give better results. In general, it seems like the NN with skip connections or the NN based on the UNet architecture should be sufficient for this form of task. The NN with skip connections also has the lowest amount of parameters, making it great to deploy on systems with low memory as well as providing the fastest inference.

5.2.3. Evaluation of the MID module

From the table 4.4 it can be seen that both the MID models from the MID module, outperform the post-processing approaches with the DICE and IoU score. This can also be confirmed in the visual evaluation of the synthetic videos, where both MIDs have an even more stable behavior than the post-processing approaches. Additionally, the predictions are overall "tighter" around the cracks, meaning they do not overextend much over the crack borders. This gives the MIDs more accurate detection results. It is, however, also the reason why both have the lowest sensitivity score (4.9).

"MID with MC" vs "MID no MC"

The MID-with-MC relies on the MC and struggles with many of the same problems as the approaches in the post-processing module. If the motion compensation is off, the resulting detection from the MID-with-MC will also be off. Also, in the rare case of an undetected MC failure, as seen in Figure 4.17, the corrupted stack can also lead to false detections by the MID-with-MC. Meanwhile, the MID-no-MC does not rely on the in this project developed MC. It instead seems to learn how to compensate for the motion itself. If that is actually what is happening is hard to know since a network like that operates as a black box in many ways. There is always the possibility that the MID-no-MC is just a SID in disguise. However, the stability in its detection result across subsequent frames suggests that it has some sort of spatial-temporal awareness.

Given the DICE and IoU scores of the SID and the MID-no-MC, they both seem to have similar performances. Based on the measured metrics, it is impossible to say which one is the more consistent detector. However, the visual evaluation indicates that the MID-no-MC seems to be more temporal stable and have less flicker.[2].

MIDs on real-life videos

The SID and both MIDs were trained on synthetic videos, which are based on different crack datasets. Unfortunately, the DNV crack dataset got a little under-represented during training. It is observed that all models (SID and both MIDs) perform extremely poorly on synthetic videos and real-life videos from inside ship tanks. Cracks in ship tanks is a more complex domain to operate on than just cracks in roads and concrete. For the detectors to work better in that domain, the ship

[2]As a result of the MID seeming to learn how to MC on its own, a DLv3NN from the post-processing module was trained, using the ground truth labels from synthetic video sequences which were not MC as input. This, however, did not work, which is not too surprising since binary masks contain little spatial information themselves.

tank dataset would need to have been more represented during training. Cracks in the ship tank dataset often are much thinner, smaller, and have a lot more appearance variation than cracks in the other datasets. Therefore specific augmentation methods such as scaling and random-crop would need to be applied to the dataset during training. Even if the detectors do not work so great in the ship tank domain, their use in the general crack domain indicates that MIDs have more temporally stable detection results than SIDs.

5.2.4. Tracking

The visual evaluation of the tracker suggests that the tracker relatively reliable manages to track cracks in-between most frames. An example of the working tracker can be seen in Figure 4.19. As long as the motion compensated stack is stable, the track's identity is stable as well, meaning there are no identity switches. However, if the motion-compensated stack resets, the tracker's identity changes. This is intended behavior since the tracker can not reliably maintain a track when the MC has failed. Since the tracker relies on the MC, it is also prone to the errors of the MC. The way the tracking module is implemented, it keeps a track alive as long as at least one pixel in the stack is identified as a crack and overlaps the track. Each frame in the motion-compensated stack is associated with a list of its containing tracks. These trackers are used to associate a detected crack in the current frame to the tracks from previous frames. This also means that parts of a tracker can be based on detections in frames that are older and outside of the stack's time window. The idea behind this implementation is to track crack pixels, even if they are not detected over a longer time period. For instance, if a crack gets partially occluded or the detector simply does not detect it for more frames than the stack is deep. However, this is where the minor errors in the MC can accumulate, making a tracker's area wider and wider over time, even if no new detection occurs inside the track. The MC stack depth is limited throughout this project precisely to prevent those MC inaccuracies from being significant. However, this implementation of the tracker bypasses this limitation, as the stack is only used for associating and compensating previous tracks. In the feature work chapter (7.3), some improvements to the tracker are suggested. There is also the problem that the tracker becomes unreliable if an undetected MC failure occurs. This often increases the area of the track. Figure 5.1 shows an example where the tracker still contains areas that are based on the corrupted information from the undetected MC failure that occurred 17 frames prior. The wrongly increased track area can make detected cracks associate with the track, which normally should not be associated.

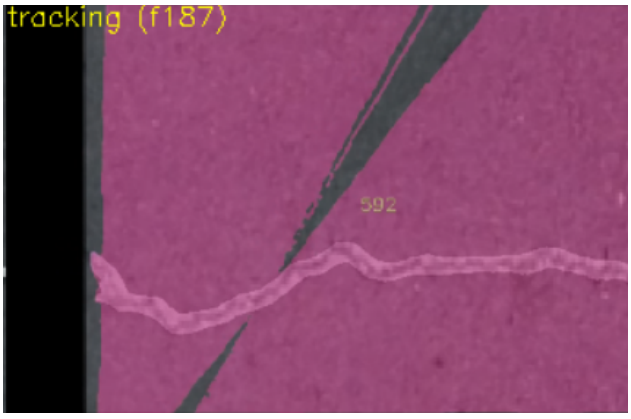


Figure 5.1.: The tracker at frame 187, 17 frames after the undetected MC error occurred 4.16. Note that the stack depth used here is 5. All of the pink pixels (light and dark) are the tracker. The lighter pink is where detection occurs in the current frame. The number 592 refers to the tracks ID

Tracking through occlusion

A crack pixel must be detected inside the tracker at least once within a stack. If this is not the case, the track is discontinued. When a tracked crack moves entirely out of the scene, the tracker is forgotten if the crack does not move back in again within the next n frames (where n is the stack depth). This is also the case if a crack gets fully occluded. However, if a crack becomes partially occluded, the tracker will still be able to track through the occluded areas[3]. Note that the tracker uses connected component analysis to figure out which detected crack pixels belong to the same crack. This is a simple form of instance segmentation. However, if there is some partial occlusion or FN detected pixels that split a crack instance in half, this method has no way to know that the two parts belong to the same instance. Of course, this is only the case if there is never a connection between the two halves.

Tracker performance

The morphological operations that close small gaps and extended the track slightly over cracks boundaries help make the tracking more reliable. The downside is that crack pixels that do not necessarily belong to the same instance can be associated with the same instance if they are too close to each other. Additionally, the minor MC inaccuracies can increase the inaccuracies of the track faster.

The more crack instances that need to be tracked, the slower the performance of the tracker is. The tracker's speed also depends on the stack depth, which in the experiment is set to $n = 5$. The stack depth determines for how many frames a track is kept alive when it is entirely occluded, or no detection occurs in the track. The tracker performance is faster when using smaller stacks, as fewer potential tracks need to be considered. However, what the best stack depth to use depends on how well a detector does and how reliable the MC is. A deeper stack keeps a track alive longer, even when there are no detections for a more extended period of time. This is great if the underlying detector is extremely conservative and has a lot of FN. However, as discussed earlier, a deeper stack makes the tracker more susceptible to the inaccuracies of the MC.

[3]An example can be seen in B.7

6. Conclusion

The main goal of this thesis is to improve crack detection in video sequences of ship tanks and track detected cracks. It is shown that by utilizing Motion Compensation (MC), information in older frames can be made available to the current frame in a video stream. This can be used to improve the temporally detection results from an already existing detector, both by using simple handcrafted and Neural Network-based post-processing approaches. Furthermore, switching from Single Image Detectors (SID) to Multi-Image Detectors (MID) improves temporal stability even more. The results of this thesis also suggest that MC does not necessarily need to be handcrafted for a MID but can also be learned by the detector itself. Moreover, the backward in time pixel-to-pixel association the MC provides, makes MC-based tracking viable.

The developed MC is based on the estimation of 2D transformations. While it generally works on planar scenes or scenes with a lot of flat surfaces, the 3D nature of the ship tank domain can lead to some inaccuracies in the compensation. Additionally, inaccuracies can occur because of rapid movement, lack of detail in frames, and other video artifacts, e.g., from compression. The MC constantly self-evaluates how well it compensates. Since many of the developed modules depend on the MC, their use is terminated if an insufficient compensation is detected. However, it is observed that in some instances, a bad MC is not always detected. This can mainly be traced back to a lot of ambiguous-looking pixels, which can be the result from a lack of detail in frames. These undetected MC errors are one of the main reasons why the metrics in this project do not line up with the visual observation of more temporally stable detection results.

RQ 1 (1.2) is about improving the detection results from an existing detector via post-processing. The voting approach shows in the visual evaluation to result in more temporally stable detections than a SID. However, minor inaccuracies in the compensation can make the pixels not line up correctly in the stack, leading to minor errors in the voting. The voting method is flexible in how many previous frames it considers (n) and how many detected pixels need to line up (λ) before a detection is accepted. These parameters can manually be finetuned to compensate for the specific errors the underlying detector has. It is observed that a higher n and low λ results in really temporally stable detections in scenarios where the motion compensation is accurate.

The NNs in the post-processing module mostly have similar results to the voting approach. As they are trained on synthetic emulated data, the models are not finetuned to work with a specific underlying detector. It is observed that the NNs behave differently when they use the output of different detectors. Given the in this project trained SID detector as input, the NNs usually overestimate. When using DNVs detector they tend to be more conservative. Using the SID on synthetic videos, most NNs outperform the voting approach. It looks like smaller networks with skip connections are sufficient, but deeper networks can be slightly more accurate, though also more conservative in their prediction. The NNs can learn to compensate for the specific behavior of existing detectors and do not rely on manual finetuning. However, since the number of input channels needs to be specified during training, they are not as flexible when it comes to stack depth as the voting approach.

RQ 2 (1.2) is about using multiple subsequent images as input into a Multi-Image detector (MID) to see if it can achieve more reliable results than a Single-Image detector. Subsequent frames might have

some appearance variation in-between them that can be used to extract various spatial information. It is observed that using a MID that takes in pre-compensated frames leads to more temporally stable detection results, and outperforms all the post-processing that operates with the same stack depth. However, this approach is also prone to the errors the MC has. The MID that learns to MC on its own seems to achieve even better temporally stable detections, which suggests that it learns to motion compensate on its own.

RQ 3 (1.2) is about the use of motion compensation in order to track a detected crack over multiple frames. This is observed to work well. However, like all the other modules that depend on the MC, it only works while the MC works correctly. The current tracking implementation holds a track alive as long as at least one pixel in the stack is detected in it. This makes the tracking especially vulnerable to inaccuracies in the MC. As these inaccuracies accumulate, the track can become more inaccurate over time. Nevertheless, as long as the detection and the MC are stable, the track has almost no identity switches. Using MC to track a detected crack is therefore feasible.

Since no video sequences with available ground truth data have been available for this project, a pipeline for generating synthetic video sequences with ground truth data from labeled crack still-images is developed. While common video artifacts such as motion blur are emulated, these videos do not have the physical properties of a real-world scene. The synthetic videos used for this project are not ideal data for training and testing the proposed models. Therefore, in a potential continuation of this project, it is crucial to have video sequences with ground truth data that either is based on real-life videos or, at the very least, have similar properties.

7. Proposals for Future Work

This chapter takes up some of the ideas on how to improve the implementation and the results of this master thesis. This includes, some ideas on how to improve the motion compensation, the overall pipeline, the tracking. At the end some ideas on how to get better training data is presented.

7.1. Improving motion compensation

The motion compensation is crucial for the performance of the post-processing module, the tracking module, and to some extend the MID module. Therefore, improving the motion compensation will also improve the performance of these modules.

7.1.1. Motion compensation performance measurement

The PSNR value used to measure the performance of the motion compensation in this project is calculated using grayscale frames. However, while this is less expensive than calculating the PSNR value between RGB frames, it is also slightly less accurate. As seen in this project, when motion compensation failures remain undetected, the PSNR value can sometimes be high, even when there are apparent differences between the frames. Likewise, the PSNR value can sometimes also be low, even when the motion compensation seems to be almost perfect. As discussed earlier, the PSNR value is especially thrown off when the frames it operates on do not have a lot of differences in the grayscaled pixel intensity value. The structural similarity index measurement (SSIM) is an alternative to the PSNR value [10]. The SSIMs design tries to model how the human visual system works. The calculation is based on the differences in illumination, contrast, and structure between frames. Alternatively, an ensemble method of SSIM and PSNR could be used to get a more robust performance measurement.

7.1.2. Motion compensation with Neural Networks

The motion compensation (MC) used in this project did not consistently achieve the desired outcome. In this project, the MID that does not get an already motion-compensated stack as input achieves temporally more stable detection results than the SID. This suggests that such a model does to some extent learn to motion compensate on its own. Therefore, it is interesting if using a separate NN for the motion compensation over the more classical computer vision approach would lead to an improved MC. If this is feasible, all the modules developed in this project could build upon the output of such an MC Neural Network.

7.2. Improving the pipeline

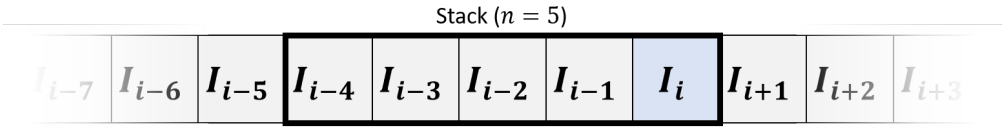
7.2.1. The ideal stack depth

A stack depth of $n = 5$ was predominantly used in the experiments, meaning that the 4 previous frames are motion-compensated to the most recent frame. The further back in time a frame is, the

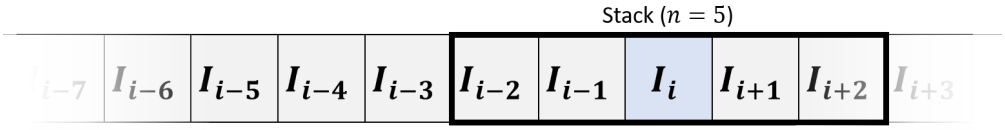
bigger the error in the motion compensation potentially becomes. This is why the stack depth is set so shallow. With a more accurate MC, a deeper stack can be used. In a hypothetical 100% accurate motion compensation, the stack depth could be infinity, meaning that every frame ever encountered in the video stream is compensated to the current frame. However, a stack depth needs to be set when using convolutional neural networks since they expect a fixed input channel size. From motion compensation to post-processing and multi-image detection, the different tasks might benefit from different stack depths. To determine the optimal stack depth for each task, experiments with different stack depths need to be conducted. For the parts involving neural networks, networks need to be trained with different stack depths and compared to each other.

7.2.2. Taking frames that are forward in time into consideration

This project treats a video sequence as a video stream. This means that the stack only contains current and older frames. However, in a video sequence also future frames are available. Therefore, if real-time detection is not the goal, both previous and future frames could be used in the pipeline. The closer frames are to each other in time, the more accurate the motion compensation usually is. Shifting the time window a stack covers to take future frames into account (like in Figure 7.1), can give the pipeline, more reliable information to work with.



(a) Currently only previous frames are included in the stack. This is because a video sequence is currently processed as a video stream



(b) In contrast to a real video stream, a video sequence also has future frames available, which could be utilized.

Figure 7.1.: Frames in a video sequence, here $I_{(i)}$ is the current frame that is evaluated.

7.2.3. Using output from previous post-processing as input

The post-processing module uses the detection maps from previous frames. In this project only the detection maps the underlying detector outputs are used. However, the detection map outputted from the post-processing in previous frames, is not used. Previous post-processing outputs incorporate information that is outside of the reach of the current post-processing (due to the time window the stack covers). Therefore, when using the output of the post-processing from previous frames in the stack, the current post-processing could get access to older information. This would enable the post-processing to recover even more lost detections. However, this requires the output of the post-processing to be reliable at each time step, since inaccuracies in the post-processing output will accumulate fast.

7.3. Improving the tracker

The tracker relies heavily on the MC. In the experiment, a stack depth of 5 is used for the tracker. The inaccuracies of the MC play a significant role in the errors that can accumulate in the tracks. As

the tracks are compensated the same way the frames are compensated for the camera motion, these errors can build up. The way the tracker is implemented, some of the pixels in a track can originate from a detection that no longer is inside the stack. This means that the tracker is susceptible to accumulated MC errors originating even beyond the stack depth. The main reason for limiting the stack depth is to prevent those MC inaccuracies from accumulating too much.

A way to improve the tracker is to only to keep track of crack pixels that originate from within the time window of the stack. This would make the tracker less prone to errors and inaccuracies in the MC. However, if the underlying detector does not manage to consistently detect a crack within that time window, the tracker would have more identity switches.

7.4. Improving the quality of training data

The performance of some parts of the project, such as the motion compensation, voting, and tracking, are independent from the available training data. However, both the models in the post-processing module and the MID module, would benefit significantly from having better-labeled videos available. With the current approach of generating synthetic videos, many details present in a real-life video are not captured or represented correctly. There has been an awareness of many of these flaws throughout this project. The necessity of better training data has become even more apparent.

7.4.1. Semi-automatic hand labeling

Two strategies of hand labeling real-video sequences were tried out in this project. The first approach was to hand label every frame in the sequence manually. For this, the open-source annotation tool, CVAT, was used. The first time a crack appears in a frame, a mask is drawn on top of it. The mask is then adjusted in the upcoming frames until the crack is not in the frame anymore. This approach turned out to be extremely tedious. Additionally, there are some variations in the masks throughout the frames, making the ground truth slightly inconsistent. The other manual approach was to mask out every n-th frame ($n = 10$) and let CVAT interpolate the mask between the frames. This approach is a little faster, but there were many inconsistencies in the interpolation. Combined with limited video sequences available, none of these approaches were feasible for this project.

However, even if more real-life videos would be available, the problem of streamlining the labeling process must still be addressed. Motion compensation can be used to semi-automate the hand labeling process. The first time a crack is seen in a frame, it is hand-labeled. When moving to the next frame, this previous frame and associated label are motion-compensated to the current frame. If more of the crack is shown or if there is an error in the compensation, the label can be adjusted. This makes the labels process overall more consistent than hand labeling each frame. It also should be better than the tracking CVAT uses. There are, however, different factors that can lead to an error in compensation. If such an error is detected, the crack needs to be relabeled. The motion compensation is not expected to be 100% accurate all the time. Like in the tracking implementation, these errors accumulate. Therefore, a threshold needs to be set for how many frames to label with motion compensation automatically before manually reevaluating the label. Even during the automatic propagation of labels with the motion compensation, this process needs to be semi-supervised.

Synthetic video generation with 3D software

In this project, synthetic videos were generating using 2D transformations applied to 2D frames. As discussed in 5.2.3, there are many assumptions made that simply are not realistic in a real-life video sequence. Additionally, elements such as changing light conditions and adding motion blur are emulated in a way that does not necessarily reflect real-world behavior. Using 3D software, video sequences and associated ground truth can be simulated to look and behave more like a real-life video.

The first step is to construct a scene with a camera and a 2D crack image, like in Figure 7.2. A light source behind the camera can be adjusted, so it lights up the 2D image unevenly. Shadows are simulated by adding 3D objects between the light source and the camera. The closer the 3D object is to the light source, the more diffuse the shadow is projected on the 2D image.

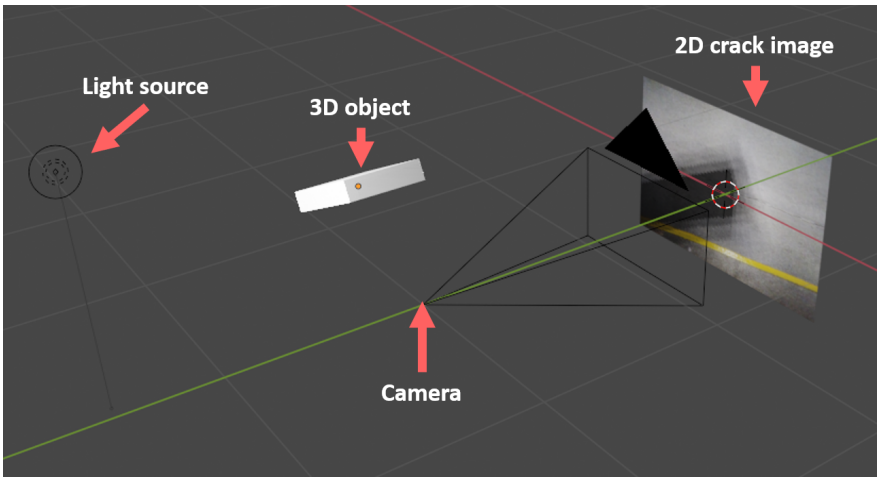


Figure 7.2.: An example of a constructed scene in using the open source 3D software Blender[1]

Like in the synthetic video approach used in this project, the camera movement can be tracked from real-life videos and then applied on the rigged scene. Many 3D programs have the option to track the camera movement in 3D. This results in an even more accurate and natural movement than in the used approach. The videos chosen as references do ideally come from the targeted domain. This means that if the expected video input mainly comes from drones, the videos used as reference should preferably also come from drones flying in a similar environment. However, a variety of movements, i.e., with handheld movement, can also be used to make the model generalize even better. When lowering the shutter speed of the virtual camera, directional motion blur will be simulated. Together with the light and the 3D objects casting shadows, the generated video becomes more realistic. Exporting the resulting crack videos with different bit rates and lossy compression algorithms reduces the gap between a generated video and a real-life video.

The crack image can now be switched out with its associated ground truth label. By turning off the impact of the light source and setting up the shutter speed to minimize motion blur, the associated labeled video can be rendered. The labels will be consistent and always accurate, given that the initial label is accurate. This is not always the case with the hand-labeled approach.

Using 3D software can even be taken a step further. A pseudo depth map can be used to simulate depth in the cracks and other structures. This makes the light interact more natural with different surfaces, leaving, among others, specular highlights. Modeling entire environments is also possible.

[1] https://www.blender.org/

This is, however, a time-consuming process and is hard to make the model look natural. The process could be streamlined by scanning real-life scenes in 3D and importing them into 3D software. However, this would make the problem even more complex and move it into an even broader field of study.

7.4.2. Using Box labels

Throughout this project, labeling and detecting cracks have been done on a pixel level. The task of pixel-level classification is, in general, more complex and computationally expensive than bounding box detection. If the goal of the detector only is to see how many cracks there exist in a given frame and track those, bounding box detection might be sufficient. It is much easier to label training data with bounding boxes than on a pixel level. A bonus is that the neural networks needed can be smaller and run inference much faster, even on lower-end hardware.

Motion compensation can still be used to post-process the detection to smooth out false positives and false negatives. The voting will work right out of the box, but the post-processing Neural Networks would need to be retained with proper bounding box training data. With some minor tweaking of the implemented tracking algorithm, motion compensation can also be used to track a detection over multiple frames. The main tweaking needed is to set some threshold (i.e., IoU, DICE) for how much an instance of a bounding box detection in the current frame needs to overlap with an existing track in order to be considered a part of it.

7.4.3. More augmentations for Single-Image detectors

The better a Single-image detector (SID) is the more temporally stable it is, when it is used on a video. The crack images from DNV have a lot of variances in crack appearance. Standard augmentations such as scaling, rotation and random cropping are intuitive to use. However, frames often have artifacts that still images do not have. Examples discussed in this thesis include motion blur and video compression artifacts. To make a SID able to detect a crack consistently it should be trained to detect a crack in images that contain such artifacts. Therefore, introducing more realistic video artifacts or labeling the frames in real-life video, should greatly improve SIDs.

Bibliography

[1] Rabih Amhaz et al. "Automatic Crack Detection on Two-Dimensional Pavement Images: An Algorithm Based on Minimal Path Selection". In: *IEEE Transactions on Intelligent Transportation Systems* 17.10 (Oct. 2016). Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 2718–2729. ISSN: 1558-0016. DOI: 10.1109/TITS.2015.2477675.

[2] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. "COCO-Stuff: Thing and Stuff Classes in Context". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Salt Lake City, UT, USA: IEEE, June 2018, pp. 1209–1218. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00132. URL: https://ieeexplore.ieee.org/document/8578230/ (visited on 06/24/2021).

[3] M. Calonder et al. "BRIEF: Computing a Local Binary Descriptor Very Fast". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.7 (July 2012). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1281–1298. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2011.222.

[4] Liang-Chieh Chen et al. "Rethinking Atrous Convolution for Semantic Image Segmentation". In: *arXiv:1706.05587 [cs]* (Dec. 5, 2017). arXiv: 1706.05587. URL: http://arxiv.org/abs/1706.05587 (visited on 06/15/2021).

[5] Konstantinos G Derpanis. "Overview of the RANSAC Algorithm". In: (), p. 2.

[6] Markus Eisenbach et al. "How to get pavement distress detection ready for deep learning? A systematic approach". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017 International Joint Conference on Neural Networks (IJCNN). ISSN: 2161-4407. May 2017, pp. 2039–2047. DOI: 10.1109/IJCNN.2017.7966101.

[7] Mark Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (Jan. 1, 2015). Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 1 Publisher: Springer US, pp. 98–136. ISSN: 1573-1405. DOI: 10.1007/s11263-014-0733-5. URL: https://link.springer.com/article/10.1007/s11263-014-0733-5 (visited on 06/15/2021).

[8] Martin Fishler and Robert Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: (1980), p. 42.

[9] Costantino Grana, Daniele Borghesani, and Rita Cucchiara. "Optimized Block-Based Connected Components Labeling With Decision Trees". In: *IEEE Transactions on Image Processing* 19.6 (June 2010). Conference Name: IEEE Transactions on Image Processing, pp. 1596–1609. ISSN: 1941-0042. DOI: 10.1109/TIP.2010.2044963.

[10] Wilko Guilluy, Azeddine Beghdadi, and Laurent Oudre. "A performance evaluation framework for video stabilization methods". In: *2018 7th European Workshop on Visual Information Processing (EUVIP)*. 2018 7th European Workshop on Visual Information Processing (EUVIP). ISSN: 2471-8963. Nov. 2018, pp. 1–6. DOI: 10.1109/EUVIP.2018.8611729.

[11] Trym Haavardsholm. "Lecture 7.2 Feature Matching". In: (), p. 18. URL: https://www.uio.no/studier/emner/matnat/its/TEK5030/v19/lect/lecture_4_2_feature_matching.pdf.

[12] C. Harris and M. Stephens. "A Combined Corner and Edge Detector". In: *Procedings of the Alvey Vision Conference 1988*. Alvey Vision Conference 1988. event-place: Manchester. Alvey Vision Club, 1988, pp. 23.1–23.6. DOI: 10.5244/C.2.23. URL: http://www.bmva.org/bmvc/1988/avc-88-023.html (visited on 12/16/2020).

[13] Jianbo Shi and Tomasi. "Good features to track". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. ISSN: 1063-6919. June 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (May 24, 2017), pp. 84–90. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3065386. URL: https://dl.acm.org/doi/10.1145/3065386 (visited on 06/15/2021).

[15] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 740–755. ISBN: 978-3-319-10602-1. DOI: 10.1007/978-3-319-10602-1_48.

[16] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 21–37. ISBN: 978-3-319-46448-0. DOI: 10.1007/978-3-319-46448-0_2.

[17] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: (), p. 10.

[18] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60.2 (Nov. 1, 2004), pp. 91–110. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: https://doi.org/10.1023/B:VISI.0000029664.99615.94 (visited on 12/18/2020).

[19] Thomas Opsahl. "Lecture 1.3 Basic projective geometry". In: (), p. 29.

[20] Thomas Opsahl. "Lecture 4.3 Estimating homographies from feature correspondences". In: (), p. 29.

[21] F. Perazzi et al. "A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV: IEEE, June 2016, pp. 724–732. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.85. URL: https://ieeexplore.ieee.org/document/7780454/ (visited on 06/16/2021).

[22] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *arXiv:1506.02640 [cs]* (May 9, 2016). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640 (visited on 06/15/2021).

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *arXiv:1505.04597 [cs]* (May 18, 2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597 (visited on 12/16/2020).

[24] Paul L. Rosin. "Measuring Corner Properties". In: *Computer Vision and Image Understanding* 73.2 (Feb. 1, 1999), pp. 291–307. ISSN: 1077-3142. DOI: 10.1006/cviu.1998.0719. URL: http://www.sciencedirect.com/science/article/pii/S1077314298907196 (visited on 12/17/2020).

[25] Edward Rosten and Tom Drummond. "Machine Learning for High-Speed Corner Detection". In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Lecture Notes in Computer Science. event-place: Berlin, Heidelberg. Springer, 2006, pp. 430–443. ISBN: 978-3-540-33833-8. DOI: 10.1007/11744023_34.

[26] E. Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011 International Conference on Computer Vision. ISSN: 2380-7504. Nov. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[27] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (Dec. 1, 2015), pp. 211–252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y (visited on 06/15/2021).

[28] Yong Shi et al. “Automatic Road Crack Detection Using Random Structured Forests”. In: *IEEE Transactions on Intelligent Transportation Systems* 17.12 (Dec. 2016). Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 3434–3445. ISSN: 1558-0016. DOI: 10.1109/TITS.2016.2552248.

[29] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv:1409.1556 [cs]* (Apr. 10, 2015). version: 6. arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556 (visited on 06/15/2021).

[30] T. Trzcinski, M. Christoudias, and V. Lepetit. “Learning Image Descriptors with Boosting”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.3 (Mar. 2015). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 597–610. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2014.2343961.

[31] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. “Simple Online and Realtime Tracking with a Deep Association Metric”. In: *arXiv:1703.07402 [cs]* (Mar. 21, 2017). arXiv: 1703.07402. URL: http://arxiv.org/abs/1703.07402 (visited on 12/18/2020).

[32] Fan Yang et al. “Feature Pyramid and Hierarchical Boosting Network for Pavement Crack Detection”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.4 (Apr. 2020). Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 1525–1535. ISSN: 1558-0016. DOI: 10.1109/TITS.2019.2910595.

[33] Lei Zhang et al. “Road crack detection using deep convolutional neural network”. In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016 IEEE International Conference on Image Processing (ICIP). ISSN: 2381-8549. Sept. 2016, pp. 3708–3712. DOI: 10.1109/ICIP.2016.7533052.

[34] Qin Zou et al. “CrackTree: Automatic crack detection from pavement images”. In: *Pattern Recognition Letters* 33.3 (Feb. 1, 2012), pp. 227–238. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2011.11.004. URL: https://www.sciencedirect.com/science/article/pii/S0167865511003795 (visited on 06/16/2021).

A. Developed models

This appendix contains the overview of the 2 models that where developed from scratch for the post-processing module. It also contains an overview of the number of parameters and estimated model size for the modified UNet based model and the DeepLabv3 based model. Here also the first layers of the architecture are also shown since, this is where the modification happen. For all the post-processing NNs the first convolutional layer takes in 5 channels. The DeepLabv3 model is also modified to be used in the MID module as the MID. In this modification the first convolutional layer takes in 15 input channels.

A.1. Small Network

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 342, 512]           2,944
              ReLU-2         [-1, 64, 342, 512]               0
         MaxPool2d-3         [-1, 64, 171, 256]               0
            Conv2d-4        [-1, 128, 171, 256]          73,856
              ReLU-5        [-1, 128, 171, 256]               0
         MaxPool2d-6         [-1, 128, 85, 128]               0
            Conv2d-7         [-1, 256, 85, 128]         295,168
              ReLU-8         [-1, 256, 85, 128]               0
         MaxPool2d-9          [-1, 256, 42, 64]               0
           Conv2d-10          [-1, 512, 42, 64]       1,180,160
             ReLU-11          [-1, 512, 42, 64]               0
        MaxPool2d-12          [-1, 512, 21, 32]               0
  ConvTranspose2d-13          [-1, 256, 41, 63]       1,179,904
             ReLU-14          [-1, 256, 41, 63]               0
  ConvTranspose2d-15         [-1, 128, 84, 127]         819,328
             ReLU-16         [-1, 128, 84, 127]               0
  ConvTranspose2d-17         [-1, 64, 170, 255]         204,864
             ReLU-18         [-1, 64, 170, 255]               0
  ConvTranspose2d-19          [-1, 1, 342, 512]           2,305
================================================================
Total params: 3,758,529
Trainable params: 3,758,529
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 3.34
Forward/backward pass size (MB): 434.47
Params size (MB): 14.34
Estimated Total Size (MB): 452.15
----------------------------------------------------------------
```

A.1.1. Small model architecture

```
SmlModel(
  (encoder): Sequential(
    (0): Conv2d(5, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
```

```
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (decoder): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): ReLU()
    (2): ConvTranspose2d(256, 128, kernel_size=(5, 5), stride=(2, 2), padding=(1, 1),
           output_padding=[1, 0])
    (3): ReLU()
    (4): ConvTranspose2d(128, 64, kernel_size=(5, 5), stride=(2, 2), padding=(1, 1),
           output_padding=[1, 0])
    (5): ReLU()
    (6): ConvTranspose2d(64, 1, kernel_size=(6, 6), stride=(2, 2), padding=(1, 1))
  )
)
```

A.2. Small Network with Skip connections

Note that the small model with skip connections has less parameters than the normal small model. This is because the small model with skip connections uses a bilinear up-scaling rather than transpose convolutions with different kernel sizes. Therefore the small network with skip connection has the fewest parameters of all the networks used in the post-processing module.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 342, 512]           2,944
         MaxPool2d-2         [-1, 64, 171, 256]               0
              ReLU-3         [-1, 64, 171, 256]               0
            Conv2d-4        [-1, 128, 171, 256]          73,856
         MaxPool2d-5         [-1, 128, 85, 128]               0
              ReLU-6         [-1, 128, 85, 128]               0
            Conv2d-7         [-1, 256, 85, 128]         295,168
         MaxPool2d-8          [-1, 256, 42, 64]               0
              ReLU-9          [-1, 256, 42, 64]               0
           Conv2d-10          [-1, 512, 42, 64]       1,180,160
        MaxPool2d-11          [-1, 512, 21, 32]               0
             ReLU-12          [-1, 512, 21, 32]               0
         Upsample-13          [-1, 512, 42, 64]               0
           Conv2d-14          [-1, 256, 42, 64]         196,864
        UpSkipper-15          [-1, 256, 42, 64]               0
             ReLU-16          [-1, 256, 42, 64]               0
         Upsample-17         [-1, 256, 84, 128]               0
           Conv2d-18         [-1, 128, 85, 128]          49,280
        UpSkipper-19         [-1, 128, 85, 128]               0
             ReLU-20         [-1, 128, 85, 128]               0
         Upsample-21        [-1, 128, 170, 256]               0
           Conv2d-22         [-1, 64, 171, 256]          12,352
        UpSkipper-23         [-1, 64, 171, 256]               0
             ReLU-24         [-1, 64, 171, 256]               0
         Upsample-25         [-1, 64, 342, 512]               0
           Conv2d-26          [-1, 1, 342, 512]              70
        UpSkipper-27          [-1, 1, 342, 512]               0
================================================================
Total params: 1,810,694
Trainable params: 1,810,694
Non-trainable params: 0
```

```
----------------------------------------------------------------
Input size (MB): 3.34
Forward/backward pass size (MB): 513.67
Params size (MB): 6.91
Estimated Total Size (MB): 523.92
----------------------------------------------------------------
```

A.2.1. Small model with skip connections architecture

```
SmlSkipModel(
  (down_conv1): Conv2d(5, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (down_conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (down_conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (down_conv4): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (down_max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (relu): ReLU()
  (up_conv1): UpSkipper(
    (up): Upsample(scale_factor=2.0, mode=bilinear)
    (conv): Conv2d(768, 256, kernel_size=(1, 1), stride=(1, 1))
  )
  (up_conv2): UpSkipper(
    (up): Upsample(scale_factor=2.0, mode=bilinear)
    (conv): Conv2d(384, 128, kernel_size=(1, 1), stride=(1, 1))
  )
  (up_conv3): UpSkipper(
    (up): Upsample(scale_factor=2.0, mode=bilinear)
    (conv): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1))
  )
  (up_conv4): UpSkipper(
    (up): Upsample(scale_factor=2.0, mode=bilinear)
    (conv): Conv2d(69, 1, kernel_size=(1, 1), stride=(1, 1))
  )
)
```

A.3. UNet based NN for post-processing

The only modification done in UNet is to change its number of input channels from 3 to 5 in its first convolutional layer.

```
(0): Conv2d(5, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
================================================================
Total params: 17,268,545
Trainable params: 17,268,545
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 3.34
Forward/backward pass size (MB): 2511.30
Params size (MB): 65.87
Estimated Total Size (MB): 2580.51
----------------------------------------------------------------
```

A.4. DeepLabv3

DeepLabv3 is used for post-processing, but also for the SID, MID with MC, MID without MC.

A.4.1. Post-processing

The modifications done in the post-processing model are to change the first convolution in the ResNet101 backbone from 3 to 5.

```
(conv1): Conv2d(5, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
```

```
================================================================
Total params: 58,632,129
Trainable params: 58,632,129
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 3.34
Forward/backward pass size (MB): 4458.23
Params size (MB): 223.66
Estimated Total Size (MB): 4685.23
----------------------------------------------------------------
```

A.4.2. SID

There are no modifications done to the DeepLabv3 model, when using it as a Single image detector.

```
(conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
```

```
================================================================
Total params: 58,625,857
Trainable params: 58,625,857
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 2.00
Forward/backward pass size (MB): 4458.23
Params size (MB): 223.64
Estimated Total Size (MB): 4683.87
----------------------------------------------------------------
```

A.4.3. MID

The modifications done to use DeepLabv3 as a MID are changing the first convolution in the ResNet101 backbone from 3 to 15. In the

```
(conv1): Conv2d(15, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
```

```
================================================================
Total params: 58,663,489
Trainable params: 58,663,489
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 10.02
Forward/backward pass size (MB): 4458.23
Params size (MB): 223.78
Estimated Total Size (MB): 4692.03
----------------------------------------------------------------
```

B. Further results and discussion

This project has access to a total of 5 real-life videos from inside ship tanks (4.1). Additional a test set of 50 synthetic videos is used in the evaluation.

B.1. About the attached videos

Attached to this thesis are some of the video results used during evaluation. Note that the best results can be observed in the result videos based on video C, D, and "Synth" as those scenes are more planar in nature than in the other videos.

- The "X_module1.mp4" videos show the different post-processing approaches in action, when run on real-life videos. All these approaches are based on the input detection of DNVs crack detector. The output of DNVs detector is displayed in the upper left corner (labeled *SID*). The grid setup of those videos are displayed in Figure B.1.
- The "X_tracking.mp4" videos show the tracking results of the different videos. The tracker uses the voting 3/5 post-processing approach as input.
- The "C_stack.mp4" video shows the voting approach with different stack depths (n) and threshold values (λ). The format of the label is λ/n. The grid setup for this video can be seen in Figure B.2.
- The "Synth_all.mp4" video shows all the different approaches run on some of the synthetic videos, using the SID (which was trained in this project) as the base detector. This is a compilation of synthetic videos, each video is run independently through the pipeline. Note that the results for the the two MIDs does not rely on the SID. A similar grid setup as used for the "X_module1.mp4" videos is used (B.1).

Figure B.1.: The grid setup for the visual evaluation of videos displaying the base detector (red) and the different post processing approaches (blue). For the synthetic videos also the MIDs results are displayed (green)

Figure B.2.: The grid setup for the visual evaluation of videos using only different voting parameters. This is the setup used for video "C_stack.mp4"

B.2. Other interesting observations

Figure B.3 shows an example where the SID does temporal worse than the MID no MC. In this particular frame (which can be seen in the "Synth_all.mp4" video), the SID does worst than most approaches (voting, post-processing NNs, MIDs).

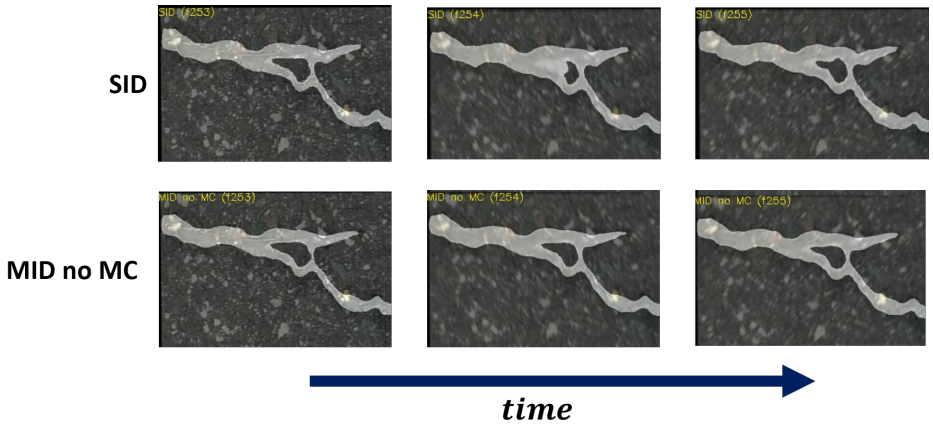


Figure B.3.: SID vs MID no MC, in frame 253 to 255 in "Synth_all.mp4"

Figure B.4 shows an example of where the MC fails completely, without the self evaluation noticing it. In this case it is observed that the voting approach outputs a lot of wrong results. The MID with MC on the other hand, does not detect anything for a while. The MID with MC is trained on motion compensated input stacks. When the frames in the stacks are not lining up with each other, the MC seems to simply not outputting anything. It is therefore clear that the MID with MC uses more than one of its input frames, to make a prediction. The MID no MC on the other hand, does not rely on the same MC. Therefore its output is stable during the whole sequence.

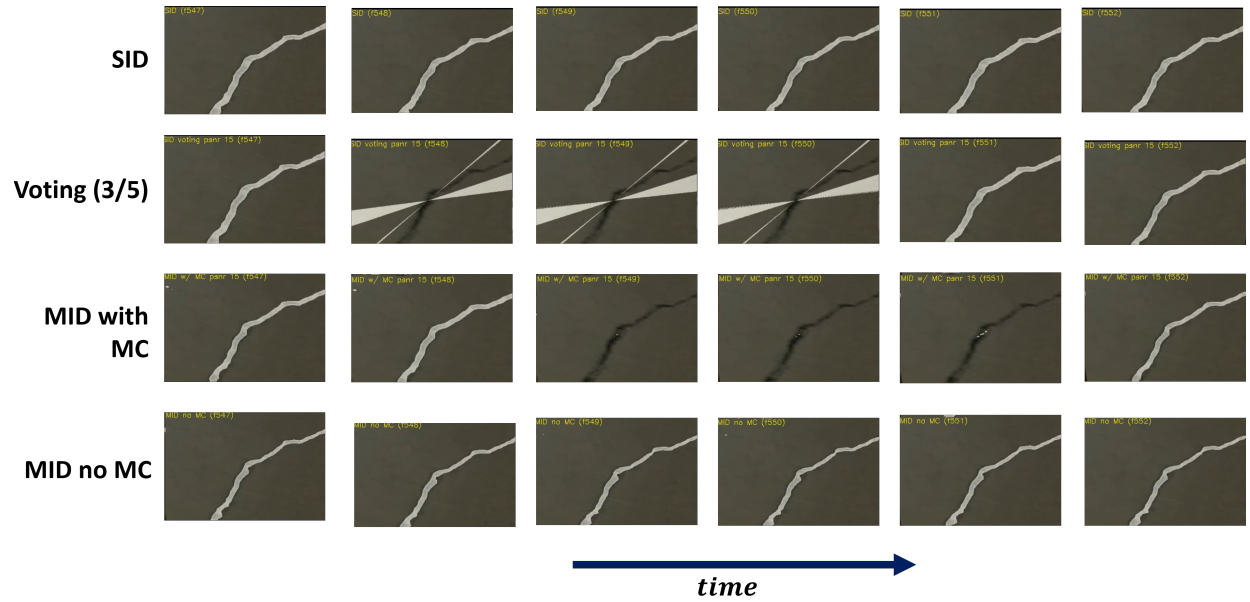


Figure B.4.: Results of different approaches when encountering a undetected MC failure. Taken from frames 547 to 552 in "Synth_all.mp4"

In Figure B.5 shows an example where using a bigger stack in the voting approach, is more susceptible to inaccuracies of the MC. Especially when looking at the voting approach 3/30 it becomes apparent that, accumulated MC errors can lead to undesirable results. This is the main reason why the stack depth needs to be limited. What the best stack depth is depends on how good the motion compensation is.

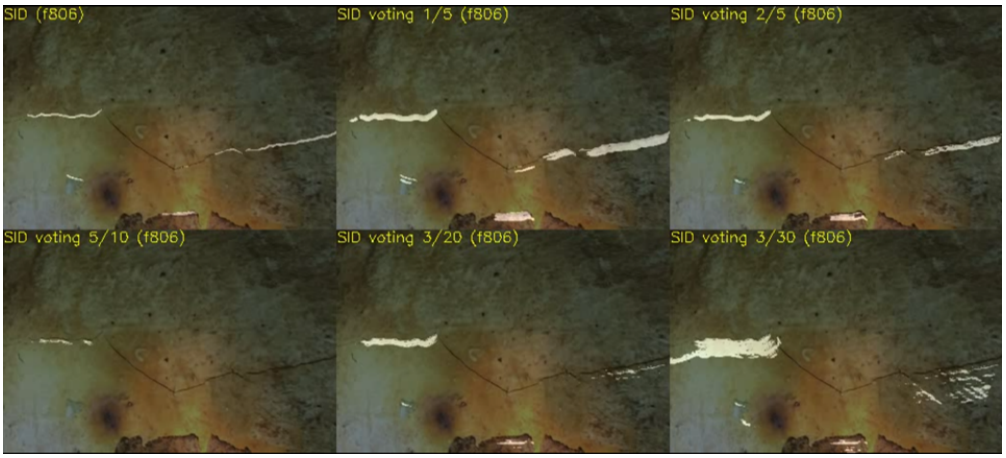


Figure B.5.: Frame 806 in "C_stack.mp4" where different parameters for post-processing by voting are used. The deeper the stack is the more inaccuracies in the MC can accumulate

A similar result can be seen in the tracking result from the same frame (B.6). While this tracker only operates on the output of the 3/5 voting approach, its track is kept around, for as long as a detection occurs inside of it. In the proposed future work chapter (7.3), solutions for resetting older tracked pixels more often is discussed. As it is implemented right now the tracker accumulates the errors from motion compensated tracks, beginning from the first frame the tracker was established. This is why the tracked area often becomes bigger over time in the examples. When looking at the entire video "C_tracking.mp4", the tracker generally works well. It also works well in the "D_tracking.mp4"

video. The other videos are set in a scene with more 3D elements. Here the motion compensation inaccuracies are greater, and therefor accumulate much faster. This is why the tracked areas in the videos "A_tracking.mp4" and "B_tracking.mp4", more often become bigger than they should be. Keep in mind that what is displayed is the track itself. The lighter colored pixels in the track are from the current detection. If the tracker would not be displayed, the tracking result would give the illusion to work properly in almost every situation.

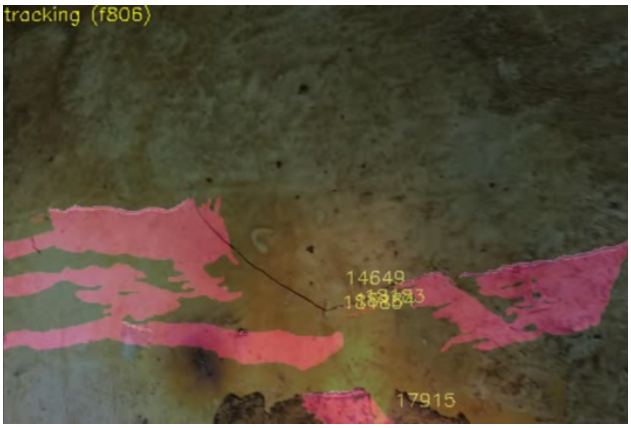


Figure B.6.: The tracks of the same frame as in B.5. Inaccuracies in the MC also accumulate in the tracks

It is also observed that the tracker can track through occlusions. Figure B.7 shows a crack that is still tracked, even while parts of it gets occluded, effectively splitting the detected crack into two halves. In this example the accumulated track inaccuracies which are discussed earlier, are also really noticeable. However, even if these inaccuracies would not be there, tracking through occlusion, would still be possible.

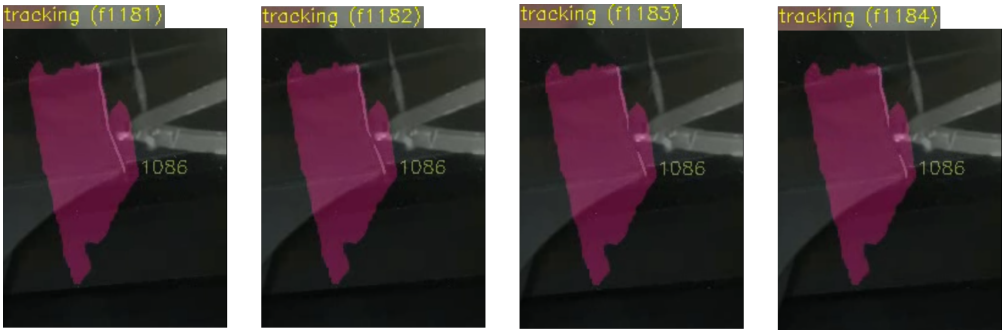


Figure B.7.: Frame 1181 to 1183 in "A_tracking.mp4". A crack can still be tracked through occlusions. The white line is the detected crack, while the red area is the track. The base video depicts a none-planar scene, therefore the inaccuracies in the track are so noticeable.

B.3. Further results for different 2D transformation types

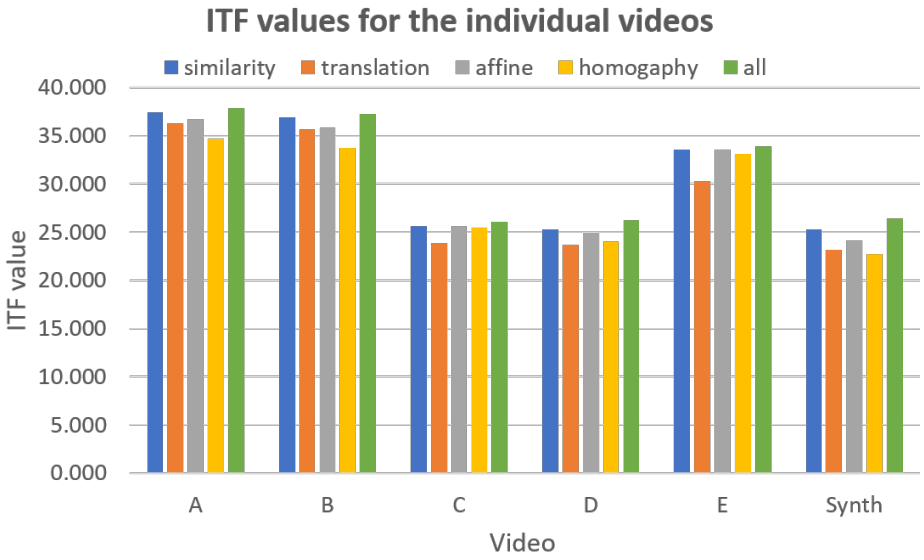


Figure B.8.: ITF values for the individual videos, when using different 2D transformation approaches. The Synth video incorporates all the 50 synthetic generated test videos. This is a breakdown of the chart in 4.3.

Looking at the results in Figure B.8, the ITF value is generally higher on the videos with a lot of 3D structures. However, when visual looking at the frames it becomes clear that this is miss-leading. The reason the ITF value is higher on A and B (despite these videos being from a drone camera moving in a scene with a lot of 3D structures) is because the overall scene is dark. This makes the PSNR value at each frame much higher, even when subsequent frames visually do not completely match. Another problem with those videos is that the drone blades are constantly visible in the frames (see Figure B.9). The drone camera seems to be on a gimbal, which tries to stabilize the footage. This mechanical stabilization makes the blades in the frame move. An assumption for the entire implementation of this project is that objects themselves are stationary. The blades visible in the camera frame however, are not stationary. The motion compensation tries to compensate for the movement of the blades and the background, which it simply is not capable of doing.

Figure B.9.: An example frame from video A. The blade structure is clearly visible in each frame

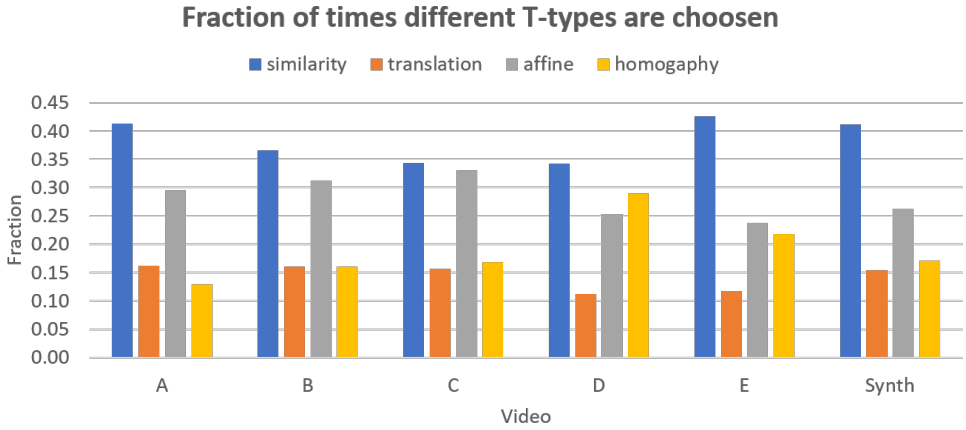


Figure B.10.: The fraction of how many times different 2D transformation types are chosen in the different videos, when run with the "all" transformation. The Synth video incorporates all the 50 synthetic generated test videos. This is a breakdown of the chart in 4.4.

B.3.1. Further results for execution time of all the approaches

The chart in B.11 shows how long the individual approaches take. Only one approach is run at the time on all the 50 synthetic test videos (2363 frames).[1]

All approaches need to be compared to the base line SID. The deeper the stack is in the voting approach, the longer it takes to run. Interestingly do the smaller post-processing NNs outperform the voting approach when it comes to execution time. The "MID no MC" execution time is only slightly higher than the SIDs giving it the least overhead.

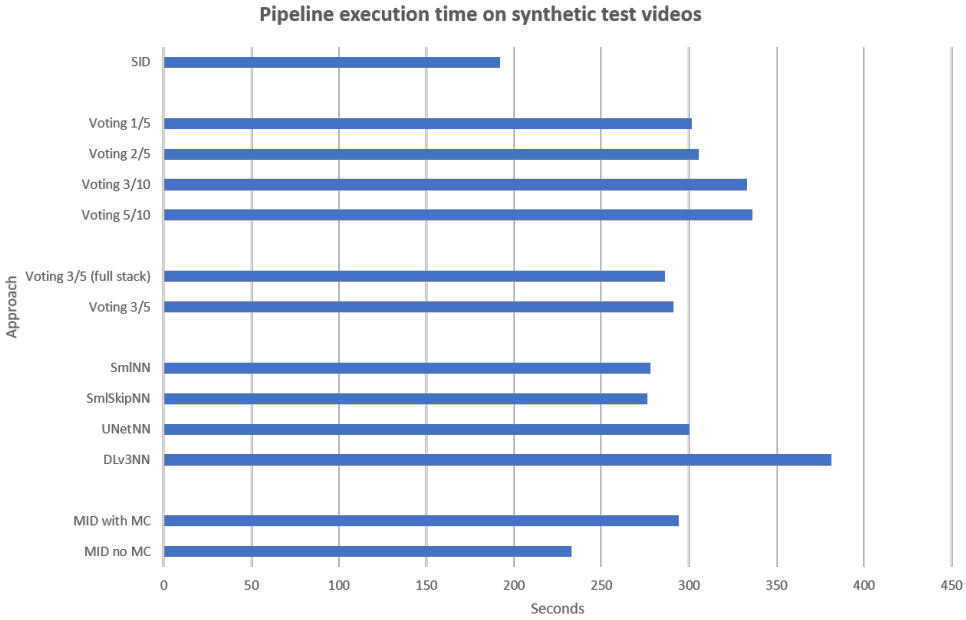


Figure B.11.: Execution time of all approaches run on 2363 synthetic frames

[1]The synthetic videos are also generated on the fly, so this also brings some overhead to the execution time

C. Numbers used for metric calculation

This is a collection of all the numbers used to calculate the sensitivity, specificity, precision, DICE and IoU in the Results chapter (4)

SID

		Predicted	
		Positive	Negative
Actual	Positive	2375	1882
	Negative	636	170212

SID voting

		Predicted	
		Positive	Negative
Actual	Positive	2062	2195
	Negative	929	169918

SID small net

		Predicted	
		Positive	Negative
Actual	Positive	3257	1000
	Negative	6889	163958

SID skip net

		Predicted	
		Positive	Negative
Actual	Positive	2885	1371
	Negative	2896	167951

SID UNet

		Predicted	
		Positive	Negative
Actual	Positive	2433	1824
	Negative	1581	169267

SID DLv3

		Predicted	
		Positive	Negative
Actual	Positive	2375	1882
	Negative	1234	169613

MID w/ MC

		Predicted	
		Positive	Negative
Actual	Positive	2220	2036
	Negative	654	170194

MID no MC

		Predicted	
		Positive	Negative
Actual	Positive	2282	1975
	Negative	473	170374

Figure C.1.: The numerical results used to calculate the confusion matrices in 4.4

SID voting 1/5

		Predicted	
		Positive	Negative
Actual	Positive	2723	1533
	Negative	2885	167963

SID voting 2/5

		Predicted	
		Positive	Negative
Actual	Positive	2322	1934
	Negative	1659	169189

SID voting 3/10

		Predicted	
		Positive	Negative
Actual	Positive	2214	2042
	Negative	1550	169298

SID voting 5/10

		Predicted	
		Positive	Negative
Actual	Positive	1990	2267
	Negative	1046	169801

SID voting stack

		Predicted	
		Positive	Negative
Actual	Positive	2079	2177
	Negative	1105	169743

Figure C.2.: The numerical results used to calculate the confusion matrices in 4.7, 4.6

D. Singular value decomposition (SVD)

SVD takes a rectangular matrix $\mathbf{A}$ of dimension $n \times p$ as an input and factorizes it into three matrices.[1] The SVD theorem is the following:

$$\mathbf{A}_{n\times p} = \mathbf{U}_{n\times n} S_{n\times p} \mathbf{V}^T_{p\times p}$$

This is also visualized in figure D.1.

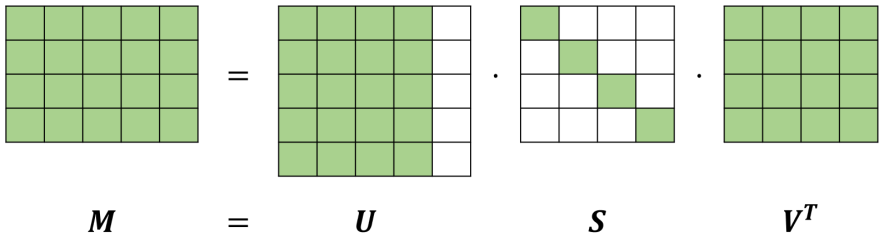


Figure D.1.: SVD of a matrix

Here:
$\mathbf{U}^T\mathbf{U} = I_{n\times n}$
$\mathbf{V}^T\mathbf{V} = I_{p\times p}$

$\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices. $\mathbf{S}$ is a real positive diagonal matrix. The entries on the diagonal of $\mathbf{S}$ are known as the singular values of $\mathbf{A}$. The columns in $\mathbf{U}$ are known as the left singular vectors, while the columns in $\mathbf{V}$ are known as the right singular vectors.

D.0.1. SVD example

$\mathbf{A}x = 0$

$$\begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Calculating SVD(A);

$$\mathbf{U} = \begin{bmatrix} -0.355 & -0.935 \\ -0.935 & 0.355 \end{bmatrix}$$

$$\mathbf{S} = \begin{bmatrix} 9.362 & 0 & 0 \\ 0 & 1.832 & 0 \end{bmatrix}$$

[1] The SVD appendix is taken from my Specialisation Project Report (Eichsteller 2020)

$$\mathbf{V} = \begin{bmatrix} -0.637 & -0.575 & -0.513 \\ 0.654 & -0.051 & -0.755 \\ 0.408 & -0.816 & 0.408 \end{bmatrix}$$

The 2 singular values from **S** are:
$s_1 = 9.362$ $s_2 = 1.832$

The 2 left singular vectors from **U** are,

$$u_1 = \begin{bmatrix} -0.355 \\ -0.935 \end{bmatrix} \qquad u_2 = \begin{bmatrix} -0.935 \\ 0.355 \end{bmatrix} \tag{D.1}$$

The three right singular vectors from **V**are:

$$v_1 = \begin{bmatrix} -0.637 \\ 0.654 \\ 0.408 \end{bmatrix} \qquad v_2 = \begin{bmatrix} -0.575 \\ -0.051 \\ -0.816 \end{bmatrix} \qquad v_3 = \begin{bmatrix} -0.513 \\ -0.755 \\ 0.408 \end{bmatrix}$$

Since there is no singular value s_3 that corresponds to v_3, the equation $x = v_3$ is considered to be a non-trivial solution of $\mathbf{A}x = 0$.

E. More about ORB

It is important to know how key points/features are extracted in each frame to understand some of the behavior of the developed Motion Compensation (MC).[1] ORB (Oriented FAST and Rotated BRIEF) is a feature detector created at OpenCV labs. The algorithm is described in the paper ORB: An efficient alternative to SIFT or SURF, 2011 [26]. SIFT and SURF are commonly used feature detectors, but they are patented, making them not free to use. The ORB paper describes how ORB is as good performing and even two orders of magnitude faster than SIFT.

Firstly ORB finds key points using FAST. In ORB, the circle's radius surrounding a potential point of interest (p) is set to 9. FAST often returns key points along edges. Because corners usually are good key points, the Harris corner measure is used to order the returned key points. The top N points are selected where N is the number of key points wanted. To obtain the scaling of those key points, ORB uses a multi-scale image pyramid. Each layer in the pyramid contains a downsampled version of the previous layer. The FAST algorithm is then applied to each layer to find key points on the different scales. Then the orientation of each keypoint is determined with a measure of corner orientation using the *intensity centroid* [24]. The moment of a shape in physics is the distribution of matter about a point or axis. In computer vision, this is quite similar, but here it is the distribution of pixel intensity. In the ORB paper, a moment of a patch is defined as:

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y)$$

The pixel intensity at a given pixel in a patch is denoted as $I(x, y)$. For binary images, this value is either 1 or 0. Suppose $I(x, y) = 0$ then this particular pixel is not counted in the sum. For grayscale images, $I(x, y)$ is a float between 0 and 1, giving each pixel different weights when summing them.

Given the above equation, m_{00} is counting every pixel in the patch that is non-zero. Assuming that the patch is binary, m_{00} represents the area of the patch. Similar m_{10} counts only all the 1's on the x-axis and m_{01} only all the 1's on the y-axis. This also works with grayscale images, but here the distribution is weighted with the intensity.

A centroid, also known as the center of mass(or in this case, the center of pixel intensity), can be determined with:

$$\mathbf{C} = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$$

It is then possible to construct a vector $\vec{OC}$ going from the center of the corners O to the centroid C. The patch orientation is then given by:

$$\theta = \mathbf{arctan2}(m_{01}, m_{10})$$

[1] The ORB summary is taken from my Specialisation Project Report (Eichsteller 2020)

Here arctan2 is a version of arctan that is aware of its quadrant. This gives the patch an orientation. Figure E.1 is an illustration of how θ is calculated.

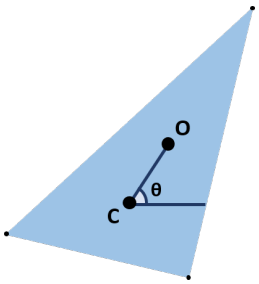


Figure E.1.: Calculation of angle θ

Given a key point in one image and a corresponding keypoint in a second image with a slightly different viewing angle, how is it possible to determine if the key points are at the same feature? A possibility is to measure the pixel similarity at the keypoint by using Euclidean distance. However, this measurement is very noise sensitive and will have problems with illumination changes, rotation, and translation between the two images.

Descriptors are used as an alternative to regular key points. They describe each key point in a way that is robust against all the image changes mentioned above. Included in a descriptor is a distance function, which is used to determine the similarity, also known as the distance, between two descriptors.

There are two common types of descriptors; patch descriptors and binary descriptors.

A patch descriptor describes the region around a key point with a feature vector. A commonly used patch descriptor is Histogram of Oriented Gradients (HOG), where the gradient of each pixel in a patch around each keypoint is calculated. HOG is used in feature detectors such as SIFT and SURF. However, this is computationally expensive.

Binary descriptors try to tackle the computational cost by encoding the information in a patch into a binary string, purely based on pixel intensity. Often the Hamming distance is used to match two binary encoded patches. The Hamming distance is the number of bit positions where the two binary strings differ. XOR is performed on the two strings, and the total number of 1's in the resulting string is counted. This is computational cheap (2.1).

ORB uses the binary description algorithm BRIEF [3] to compute a descriptor for every keypoint found with FAST. Each descriptor is converted to a binary feature vector which is 128-512 bits in length, containing only 1's and 0's. To make the descriptor selection less sensitive to noise, the image is smoothed with a Gaussian kernel.

After that, BRIEF does the following n times for every keypoint, generating a n-bit vector. Firstly two random pixels in a patch of a given size surrounding the given keypoint are selected. The first pixel is located randomly with a Gaussian distribution with a standard deviation of σ centered at the key point. The same thing is done for the second pixel, but this time the Gaussian distribution centers the first pixel, and the standard deviation is $\frac{\sigma}{2}$. If the first pixel is more bright than the second pixel, the corresponding bit is set to 1, else it is set to 0. This binary test τ on a patch p of size $K \times K$ is defined the following way:

$$\tau(p; x, y) = \begin{cases} 1, & \text{if } p(x) < p(y) \\ 0, & \text{otherwise} \end{cases}$$

where $p(x)$ is the pixel intensity of point x, and $p(y)$ is the pixel intensity of point y in the patch. To generate the binary feature vector of length n, run n binary tests on n pixel pairs:

$$f(n) = \sum_{1<i<n} 2^{i-1}\tau(p; x_i, y_i)$$

It becomes apparent that the rotation of key points is not taken into account when using BRIEF. In fact, the matching of key points falls short even with a few degrees of change in-plane rotation.

Therefore the creators of ORB created rotation-aware BRIEF (rBRIEF), adding rotation awareness while keeping the speed of BRIEF. While BRIEF chooses the sampling pairs by random, ORB learns which pairs are the most optimal to use.

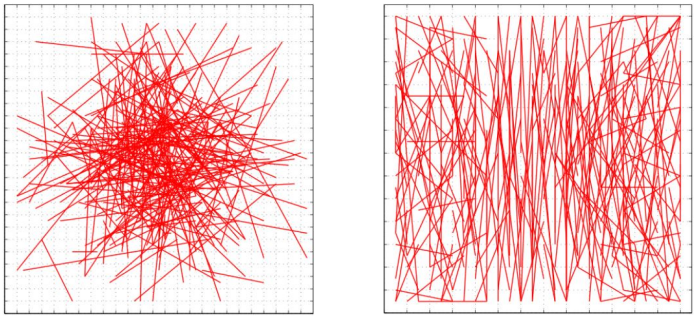
Given the orientation of the key points, ORB "steers" BRIEF accordingly. For any set of pixel pairs in the binary test, $\tau(p; x_i, y_i)$ a $2 \times n$ matrix is defined. This matrix S contains the orientation of the patch θ. The corresponding rotation matrix R_θ is found, and a steered version of S, S_θ, is constructed.

$$S_\theta = R_\theta S$$

ORB then converts the angle to an increment of $2\pi/30$ (12 degrees), and a lookup table with precomputed BRIEF patterns is constructed. When the orientation of a patch θ is consistent between two different views, the correct set of points S_θ will be looked up and used to compute the corresponding keypoint descriptor.

A sampling pair should, if possible, be uncorrelated. Every new sampling pair would then bring new information to the table. Another property the sampling pairs should have is high variance. This makes each pair respond differently to different inputs, thereby making the capture feature more distinct. By squeezing more specific information into a descriptor, the amount of information each descriptor carries becomes maximized.

In the ORB paper, it is described how the authors learned sampling pairs with those two properties (E.2). They calculated that a patch of a given size contains around 205 000 possible binary tests. Only 256 of those tests should be chosen. After gathering around 300 000 key points from the PASCAL 2006 dataset, they ran a greedy algorithm to find the 256 points that are relatively uncorrelated and with a high variance. This means that ORB uses learned, rather than random sampling pairs, when selecting pixels in a patch.



(a) Random sample pair selection with BRIEF

(b) earned sample pair selection with ORB

Figure E.2.: Orb sampling pair selection (taken from trzcinski 2015 [30])

In general, ORB outperforms BRIEF in affine transformations, as BRIEF is not designed to handle changes in rotation and scaling. This is why ORB is chosen for the motion compensation in this project . Interestingly does BRIEF outperform ORB slightly when there is a significant perspective change between two images. It also performs better in non-geometric transformations, i.e., exposure and blurriness.

F. Video presentation

Attached to this thesis is a video presentation ("Presentation.mp4") about this project. The presentation was shown at the Norwegian Open AI Labs Partner Assembly (February 2021) and at the AI Lab Master Thesis Pitch Event (Mars 2021). It is about 5 minutes long.

The video contains an overview of my specialization project and covers the main ideas for this master thesis.

NTNU
Norwegian University of
Science and Technology

DNV