PREFACE

The language of this report is UK English. Natural numbers from one to nine will be written out in letters, while numbers above nine will be written numerically. An exception to this is where the numbers are of high statistical importance, in which case also the numbers from one to ten will be written with numbers.

References will follow a variant of the AMSalpha standard [AMSalpha] developed by the American Mathematical Society. That means that references will be within square brackets and contain the author's last name or an abbreviation, followed by the year published if this is available. If the author has multiple releases within one year then the alphabetical letters will be used. Examples of this are [Blinn79] and [Foley95a].

All referenced papers and websites can be found on the accompanied CD in the *References* folder. Internal references will be enclosed in italic, like this: "1. Introduction".

When external images are used, the name of the source will be written in parenthesis in the figure description.

File and folder names as well as code or mathematical expressions will be written in *italic*.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF CODE BLOCKS

1. INTRODUCTION

Traffic safety is a priority of Danish politicians and has been so for many years [Komm09], and this focus has led to improvements in traffic safety. However, despite of these initiatives, in 2008, 406 people were killed in Denmark, 2831 were seriously injured, and 3092 were slightly injured [Statistik08a].

By logical reasoning it makes sense to assume that a considerable amount of these accidents occur when a road user collides with one or more objects. This could be a vehicle colliding with a static object or two vehicles colliding with each other. The human eye is fed a very large amount of impressions and many of these are filtered out leaving only some of the impressions to be processed at a higher cognitive level [Wolfe09].

This means that if a road user is not paying attention or gets distracted for just a moment he can miss vital information which could be crucial in preventing a collision. It also means that if it is possible to prevent collisions, it would be easy to draw the conclusion that there will be a decrease in the number of killed and injured in the traffic.

One approach to this problem is by using computer vision to allow computers to see where humans can not or react where humans are too slow. There must be an accident situation that computer vision can help prevent or an object that could thrive from being recognized automatically. This leads us to this project: We want to take a stab at saving lives in the traffic by the use of computer vision.

1.1 INITIAL PROBLEM STATEMENT

Based on the motivation in *1 Introduction* the following initial problem statement is agreed upon:

"How can one prevent collisions in traffic accidents by use of computer vision?"

The magnitude of the problem will be looked at in the pre-analysis. Statistics will be looked at to shed light on who are the most vulnerable, and what the more common collision-related accidents are. It is also relevant to look at state of the art in the field of traffic safety, and what is being done to prevent collision between vehicles and between vehicles and Vulnerable Road Users (VRU)[1]. To limit the focus we will only look at state of the art involving computer vision.

[1] Pedestrians, cyclists, moped riders, motorcyclists, etc.

2. PRE-ANALYSIS

The purpose of the pre-analysis is simply to go from a state of motivation for solving a loose and wide problem to define a specific problem which is possible to test.

As stated in the initial problem statement the task is to *prevent* collisions. But there are a lot of different traffic collisions involving different parties and circumstances. An obvious question to ask is which kind of collisions happen? Is there a way to classify the collisions in order to do a qualified selection amongst them, or maybe they are so alike that the solution to prevent them is the same in all situations? This chapter will try to get a better understanding of these questions in relation to this project.

2.1 PARTIES INVOLVED IN TRAFFIC ACCIDENTS

All collision accidents can by logical reasoning be divided into these three sections:

- Solo accidents involving only one road user
- Two-party accidents
- Multiple-party accidents

There may be major differences between the cause of accident of the three sections. It is reasonable to believe this would also result in very different solutions.

If only the initiating part of a multiple-party accident is studied it can in principle be regarded as a two-party accident, as the collision has to start with a collision between two parties and then becomes a multiple-party accident when more road users get involved. This is the reason why multiple-party accidents are considered to be a complicated kind of two-party accident and will not be independently elaborated on further.

Looking at statistics from Statistics Denmark as seen in Figure 1 it is apparent that more than twice as many accidents with two parties occur compared to solo accidents.

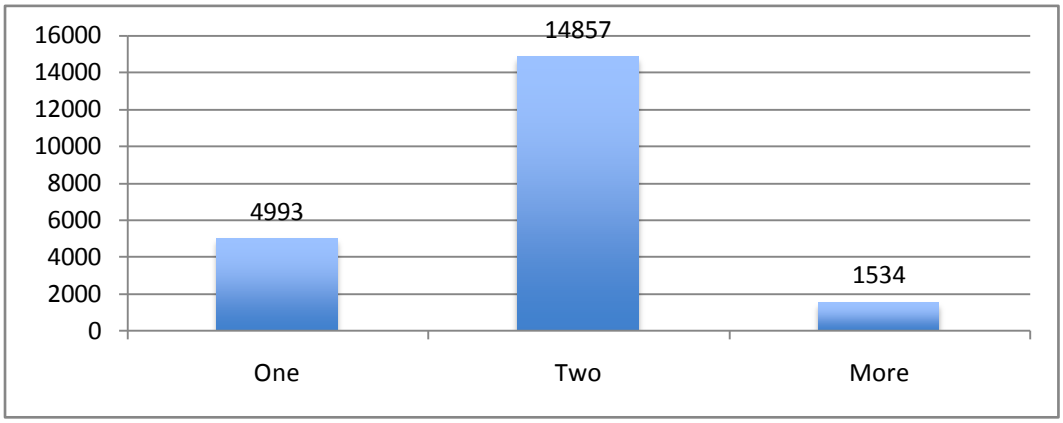


Figure 1: No. of parties involved in traffic accidents (2005-2008) [Statistik08a].

Therefore it makes sense to look at two-party accidents.

2.2 SITUATIONS IN TRAFFIC ACCIDENTS

In order to select a specific situation where collisions occur a number of choices have to be made. This process follows a chronological order so that the most general choices are made in the beginning, ending up with decisions between a couple of narrow possibilities. The line of choices before ending with a specific kind of collision is:

- The choice between collisions between two vehicles and collisions between vehicles and VRUs.
- The choice between specific vehicles or specific VRUs .
- The choice between different standardized accident situations within the already chosen categories.

The choice between collisions between two vehicles and collisions between vehicles and VRUs are primarily made in the light of earlier computer vision-related initiatives taken within the two areas. An argument for choosing one over the other is if there is a difference between the extent of research done and products made within the given area. There will therefore be taken a look at the state of the art projects in "2.2.1.1 Collisions Between Vehicles" and "2.2.1.2 Collisions Between Vehicles and VRUs" in the sections below.

We choose not to focus on collisions specifically between VRUs and other VRUs as a look at statistics shows that nowhere near as many people are injured and killed in collisions between VRUs, as seen in Figure 2.

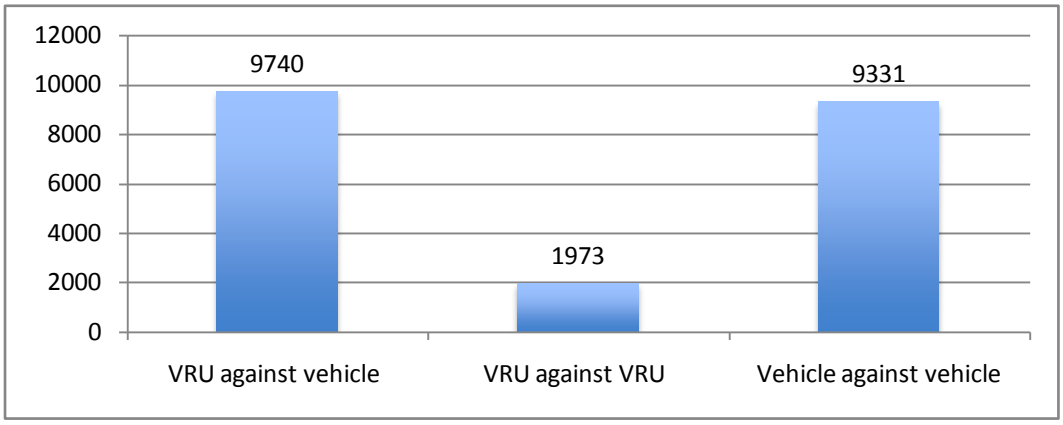


Figure 2: Injured and killed in traffic accidents (2005-2008) [Statistik08a].

2.2.1 STATE OF THE ART

Today a number of technologies which can prevent collisions using computer vision have been developed and used by vehicle manufactures. In order to gain knowledge and get inspired, these technologies will be researched and described. This next section will look into initiatives and solutions made towards decreasing the amount of collisions between vehicles. Following will be a section on the initiatives done to improve traffic safety between VRUs and vehicles.

2.2.1.1 COLLISIONS BETWEEN VEHICLES

The Intelligent Car Initiative [Commission06] by the European Commission has put up a framework of solutions in which they are working towards a 50% reduction of traffic fatalities from 2001 to 2010 on European roads. Solutions to inform the driver and keep the car on the road are made and are used in many newer cars. Many of these systems are designed to reduce collisions, fatalities and ensure a safer environment. Investments to increase the research, development and deployment of such solutions have been made since 2001 and today systems are monitoring, informing and guiding the drivers in various ways.

The Intelligent Car Initiative has dedicated itself to work towards a world where cars do not crash. In the past eight years over 400 million Euros have been invested in research, development and deployment of intelligent systems, to increase the safety and reduce the amount of accidents happening on the European roads [Commission06]. It is therefore relevant to take a look at what initiatives they have been involved in.

The Intelligent Car Initiative works as a framework combining activities related to intelligent auto mobiles. They have listed a wide range of technologies and systems which are in use today or under development. Some of

those used today are Anti-lock Braking System (ABS) and Electronic Stability Control (ESC), and other techniques such as obstacle and collision warning systems are under development or already available.

A number of different techniques to prevent vehicle vs. vehicle collisions exist, one is a solution to prevent cars deviating from their lanes with the risk of colliding with a nearby car or an obstacle along the road. Solutions preventing cars from deviating from their lanes are today used by Audi, Volkswagen, Citroën, and Lexus among others [Kohoutek08].

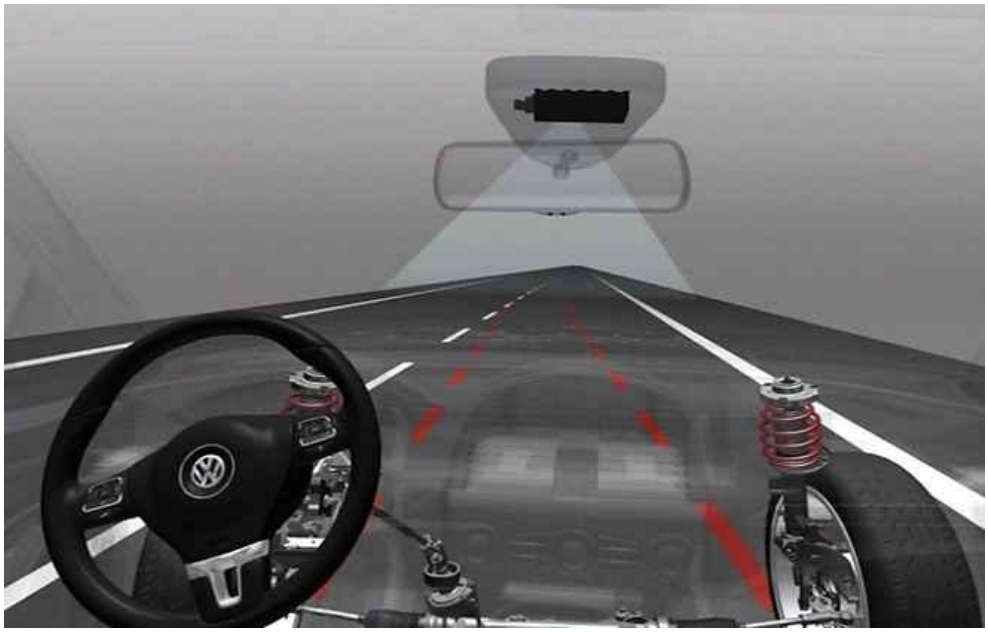


Figure 3: Lane Assist by Volkswagen (Volkswagen).

In general these solutions involve image processing by having a camera take images of the road while a computer runs filters that determine the lane markers on the road in real time. These systems can then inform or alert the driver, either visually or with vibration; some of them can even take control of the steering and correct the direction if the car is about to deviate from its lane.

Issues with cars involved in accidents when changing lanes or when the driver overlooks another car positioned in a blind spot are addressed by having two sensors monitor the rear blind spots of the vehicle [Qiu08].

With the solution from [Qiu08] the driver is warned visually with a signal in the mirror and discreet vibration in the steering wheel, if a moving obstacle is located in the blind spots on the car, when the driver is about to leave his current lane, as seen in Figure 4. The system is activated when the driver activates the turn signal [Qiu08].

Figure 4: Signal warns driver if a car is in the blind spot (Smartmicro).

It can be concluded that technology to prevent cars from deviating from their lanes exists in several variants. Another type of collision is front-to-back collisions. When looking at this kind of collision, a system called Collision Warning System has been created to increase driver awareness and warn the driver about overlooked obstacles or a possible collision with another car.

A warning is essential when the driver is distracted or overlooks a possible risk. When implementing this system it is often combined with other solutions that utilize radar or laser-based technology, which is useful to prevent rear-end collisions, e.g. when parking the car. In such cases the system is called park assist or reverse assist, as seen in trucks or newer cars [Kohoutek08]. This can be characterized as a system that protects material possessions, which is not the focus of this project and will probably not be looked at further.

When driving in dense traffic or at high speed behind another vehicle, collisions may occur if the car in front suddenly brakes and the driver in the vehicle behind does not react fast enough. A solution to these situations is to inform the driver and automate an emergency brake for the driver if there is no reaction to the warning given. Such systems are called Adaptive Cruise Control (ACC) and Active Brake Assist (ABA) as seen in Figure 5 [Schneider05].

Figure 5: Active Brake Assist (Ford)

The intention with ACC and ABA is to automatically adjust the speed and distance according to the vehicle ahead of you and automatically keep a safe breaking distance.

Closely related to Adaptive Cruise Control are speed alert systems that constantly inform the driver when the speed limit is reached. Speed alert is helping the driver to have awareness of his current driving speed which decreases the risk and consequences of having an accident occur. Speed alert can function by GPS or by having a camera record and read speed limit signs, and if the driver exceeds the posted speed limit the driver is alerted [Marsden00].

The technologies looked at so far works regardless of what time of the day it is. There are also techniques which are specifically designed for night time driving. When it comes to aiding drivers driving while it is dark or while on long trips, systems such as Night Vision Enhancement and Drowsiness-detection exist. Night Vision Enhancement systems work by having an infrared camera, located in the front grill of the car, detect infrared

light waves. The footage is transmitted to a monitor located in the dashboard. This gives the driver an overview of the road ahead when driving at night [icar06].

To prevent drivers from falling asleep when driving, the Drowsiness-detection system works by having a camera target the face of the driver, focusing on the eyes. Here infrared light is used to light up the drivers eyes and make them shine brightly for a camera in order for it to be able to track the eyes. Infrared light falls outside the spectrum of light visible to the human eye thus the driver is not affected by the light [Wolfe09]. The timeframe between each blink is monitored and the system warns the driver with light, sound or a vibrating steering wheel if the driver's eyes remain closed too long [Sayed09].

The car industry seems to take great effort in protecting their customers. If you buy a car equipped with one or more of the mentioned technologies offered by the various car manufacturers it is reasonable to assume you are more likely to avoid collisions with other vehicles. With all the presented technologies combined the car appears to be protected against collisions all around the car. Further investigations of vehicles colliding with vulnerable road users will be made.

2.2.1.2 COLLISIONS BETWEEN VEHICLES AND VRUS

The following section will look at initiatives attempting to decrease collisions in accidents where one part is a Vulnerable Road User.

The Danish Road Safety and Transport Agency (RSTA) has a focus on sensor technology regarding right-turn accidents [RSTA08]. They mention a major project called SAVE-U [SAVEU05] as one of the forerunners in the field. Since the RSTA is the authority on road safety in Denmark which makes their conclusions quite relevant. The purpose of the SAVE-U project has been to develop an integrated safety concept in vehicles to protect VRUs.

The SAVE-U project used both radar sensors, IR and video cameras. At the end of the project, in 2005, the conclusion was that the system was not good enough to be used in practice. RSTA also stated that the developed system, and all similar systems worldwide, were not able to identify all the VRUs that they should have detected. The best system detected 9 out of 10 VRUs and less than 1 out of 10 detections were there was no VRU, which was not deemed acceptable. One argument for this is, that a sensory system, that doesn't detect all VRUs, would give the driver a false feeling of trust. RSTA also stated that they expected that optimal sensory systems would be developed in 3 - 10 years. It is reasonable to believe that since the release of the 2005 report progress has been made within the field both in regards to research and technology, which makes it likely that an improved system already exists. The following sections will look into the latter part.

A survey from 2007 [Gerónimo07] summarizes 11 related state of the art projects concerning systems designed to protect pedestrians. Because of the groundwork already done in this survey, the following sections will be based on the information in it. The survey states that in the field of pedestrian protection systems using computer vision, two different lines of work are being researched: One based on images of the visible spectrum (daytime) and one based on thermal infrared (mainly for night time).

The survey argues that no comparative framework existed for comparing these protection systems. The report proposes a common framework based on the main subtasks of pedestrian detection, and used to give an overview of the techniques and main challenges for the future. To compare the 11 projects, a list of modules covering the parts of pedestrian detection systems, using computer vision, is used. They are:

- Preprocessing - obtaining original image and applying minor adjustment such as gain.
- Foreground Segmentation - isolating targets.

- Object Classification - classifying pedestrians as pedestrians.
- Verification and Refinement - verifying that pedestrians are in fact pedestrians.
- Tracking and Applications - alarm system, etc.

The report concludes that the methods used in each module must still be improved before one can expect convincing results. It continues by describing strengths and weaknesses for various techniques.

First, the report looks at techniques used for Foreground Segmentation. Strengths of Foreground Segmentation based on binocular stereo (allowing distance measuring with two cameras) are: Robustness to illumination changes and provided distances are useful to determine regions of interest at the foreground segmentation itself, for tracking and as associated information of the detected pedestrians. Drawbacks are high computation times and problems when uniform areas appear. Despite this, the report still concludes that binocular stereo is a reliable technique. Another technique, rough appearance, is, according to the report, not so promising, since it doesn't seem sufficient by itself.

The report then looks at techniques for Object Classification. The first technique is silhouette matching which matches objects with a head-shoulders-like pattern to identify a pedestrian. Finer template matching is also possible. However, the report quickly concludes that silhouette matching is not sufficient. It relies on an appearance-based application called feature extraction which is a method able to identify pedestrians by comparing features with existing examples of pedestrians or non-pedestrians. However, the report also concludes that there is a lot of work to be done in this field as well.

The next step is verifying if all the objects are in fact pedestrians. A lot of projects use previously mentioned techniques, for example by combining them. One suggestion is also to analyze the gait pattern for a pedestrian walking perpendicular to the camera, which of course requires that the target is first identified as being a pedestrian. A tracking filter called the Kalman filter is very popular due to it being able to predict the trajectory of a moving object effectively. Finally, the report states that it is hard to determine which is the better approach, because of the constant improvements within all the fields.

Several recent reports go into great detail in explaining different approaches to detect VRUs [Gandi08, Gavrila06, Jung09]. They explain how to use silhouette matching, motion-based detection, depth segmentation using binocular stereo and much more. The reports are too detailed to dissect at this point, but they seem very useful for later use.

A lot is being done in the field of VRU protection systems, but it does not seem like any of the systems have been a huge success. Since these systems are built to save lives and prevent injuries, they have to live up to a very high standard. As earlier mentioned the SAVE-U project achieved a success rate of detecting 9 out of 10 VRUs, which was not deemed effective enough. A lot of projects have potential but many implemented systems seem to suffer from lack of computational power for the calculations needed or just some minor improvements.

Summary

Compared to collisions between vehicles it seems there are less working initiatives when it comes to preventing collisions between vehicles and VRUs, even though something is still being done. At this point it is not clear which of the two that need additional attention since there are several initiatives in both cases. It is now relevant to look into what statistics can tell us about the frequency and differences between the two kinds of collisions.

2.3 COLLISION STATISTICS

It has now been looked at what is being done to prevent collisions between the involved parties, and it is clarified that more focus is directed at protecting occupants in collisions between vehicles than the other case. To be able to narrow the area of interest further down and reach the final problem statement it is now relevant to look into what statistics tells us about these collisions: How many people are killed and injured in the two cases?

During the following section statistics concerning the amount of collisions in the Danish traffic will be gathered and interpreted. These statistics cover the difference between accidents with injuries or a fatal outcome in the situation with two vehicles colliding versus the situation with a vehicle colliding with a VRU.

It is important to note that the statistics about traffic from Statistics Denmark only include injuries reported by the police. In order to examine the so-called underreporting of figures Statistics Denmark has since 1996 conducted a study where data on persons treated by casualty wards have been included. The studies have shown that the total number of injuries is almost five times higher than the number registered by the police. However, the coverage with respect to persons killed is almost 100 per cent [Statistik08b].

As seen in Figure 6 the difference between the number of fatally injured persons in the two situations are rather small. There is a tendency towards more fatal injured in the vehicle against VRU situation with a total of 566 persons reported dead in the period 2005-2008 compared to 515 dead persons in the situation involving vehicles colliding with vehicles.

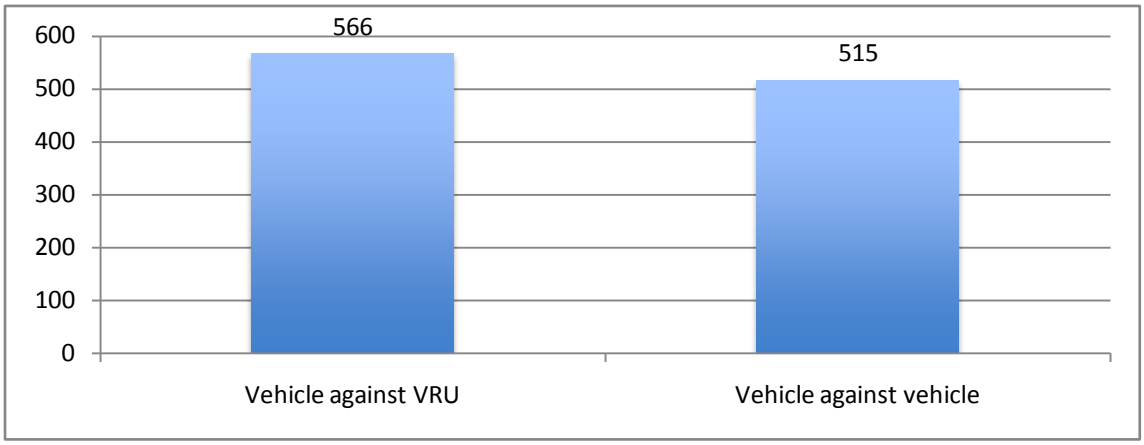


Figure 6: Fatally injured in traffic accidents (2005 - 2008) [Statistik08a].

Somewhat the same pattern is revealed when it comes to the numbers of slightly and seriously injured persons (See Figure 7). The tendency leans towards more people being injured in vehicle against VRU situations.

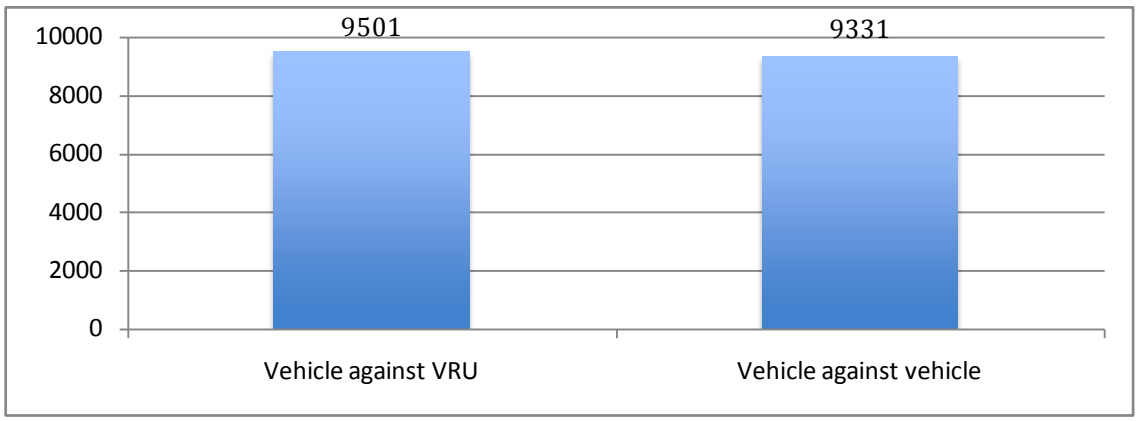


Figure 7: Slightly and seriously injured in traffic accidents (2005 - 2008) [Statistik08a].

It is important to consider the amount of the different groups in the traffic. According to an initiative by the Danish Road Safety Council [Sikkertrafik07] the risk of getting injured or killed is six times larger per driven kilometre when riding a bicycle compared to driving a car.

The development of people killed in road traffic has been on a somewhat steady decline from 1998 to 2007 where it seemed to rise again, see Figure 8. Compared to the decrease of 2% of the number of people killed as pedestrians, cyclists, moped riders and motorcycle riders it seems like the safety for VRUs has not changed significantly the past 10 years, while the safety for car drivers seem to have improved (even with two years of increasing deaths in 2006-2008).

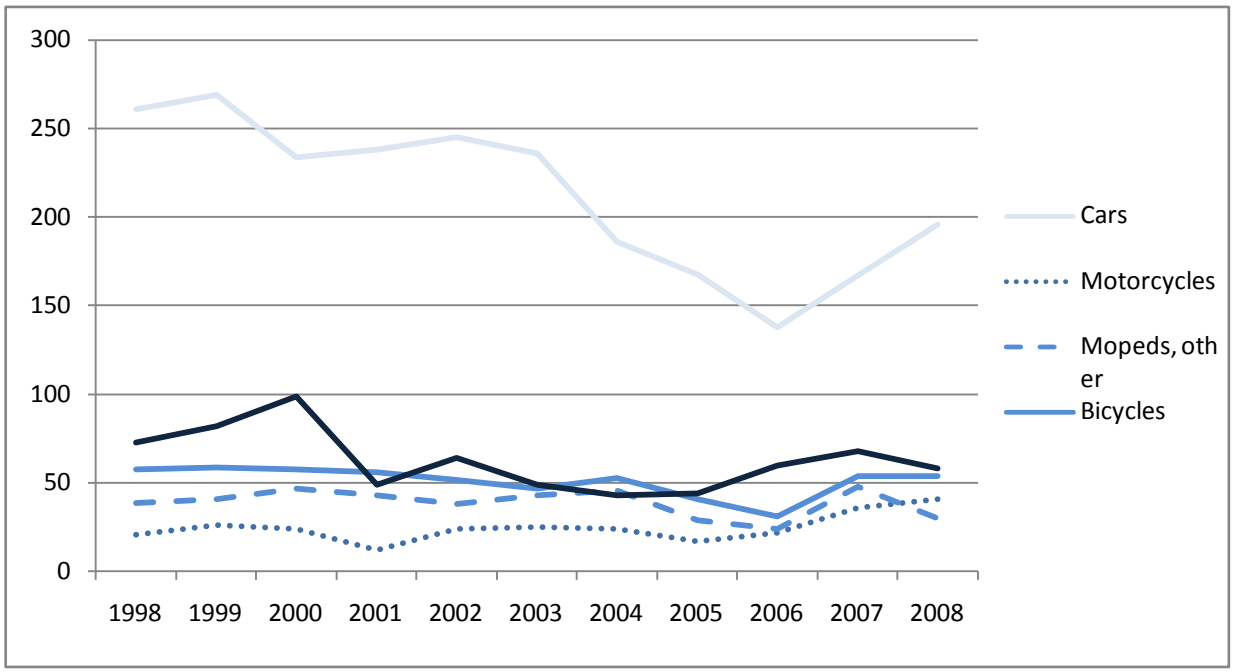


Figure 8: Deaths in traffic accidents listed after means of transportation [Statistik08a].

Based on what we have learned from statistics and further backed by state of the art where a clear difference between successful initiatives was shown, it is decided to narrow the focus down to only include collisions between vehicles and VRUs. In the statistics covered, although most people get killed in accidents with two vehicles involved, it seems interesting to research further in a field where less initiatives are taken. At the same time it is important to remember that it is far more risky to go by for instance bicycle than by car measured in distance covered.

Choosing specific vehicle and VRU

The next step is choosing between specific vehicles and specific VRUs, or simply choosing not to focus on a specific group. Here the statistics about the Danish traffic will help to make the right decision. The differences amongst the VRUs is first to be considered.

When looking at the number of injured and killed amongst VRUs (see Figure 9) it is apparent that bicycles have a consistent lead over mopeds and pedestrians. The motorcycles and moped 45s have a relatively low casualty rate compared to bicycles, mopeds, and pedestrians. The statistics would suggest that since most of the injuries are sustained by bicycles, mopeds and pedestrians, they would be the wiser choice to focus on.

It has to be noted that the graph in Figure 9 covers all kinds of personal injuries including lone accidents. It has not been possible to exclude lone accidents from the statistics.

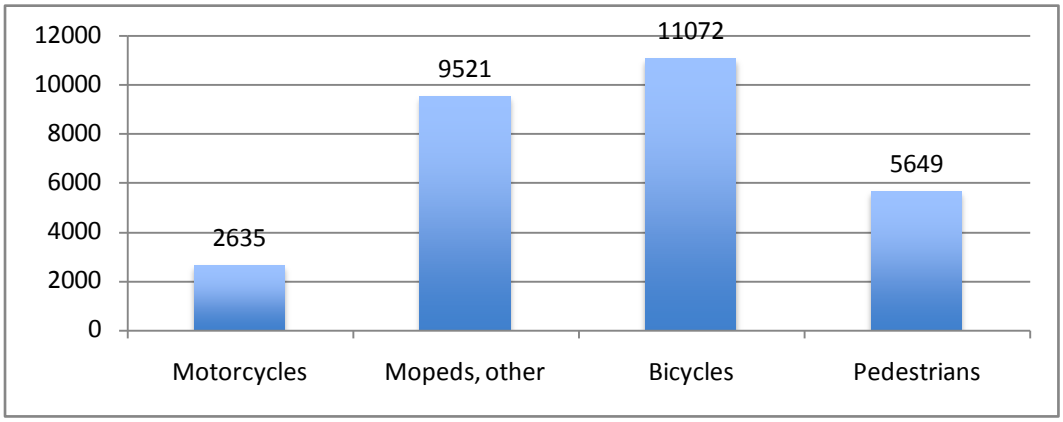


Figure 9: People injured and killed in traffic accidents listed after means of transportation (1998-2001) [Statistik08a].

Considering the fact that bicycles are consistently in the lead it makes sense to choose them over the rest. It could be argued that since bicycle and mopeds share many properties in traffic (e.g. occupying the same part of the road) and mopeds rate high on the graph that mopeds should be included as well. Furthermore, there is reason to believe that any product able to detect bicycles should be able to detect mopeds also. However, at this point we choose to focus on bicycles only for the simple reason that it will not only keep the project very specific, it will also simplify the final test scenario.

At this point it is relevant to look at which vehicle(s) to focus on. When looking at what kind of vehicle have been involved in collision with cyclists in a period of eight years between 2001 till 2008, it is apparent that cars have an obvious lead over the rest, as seen in Figure 10.

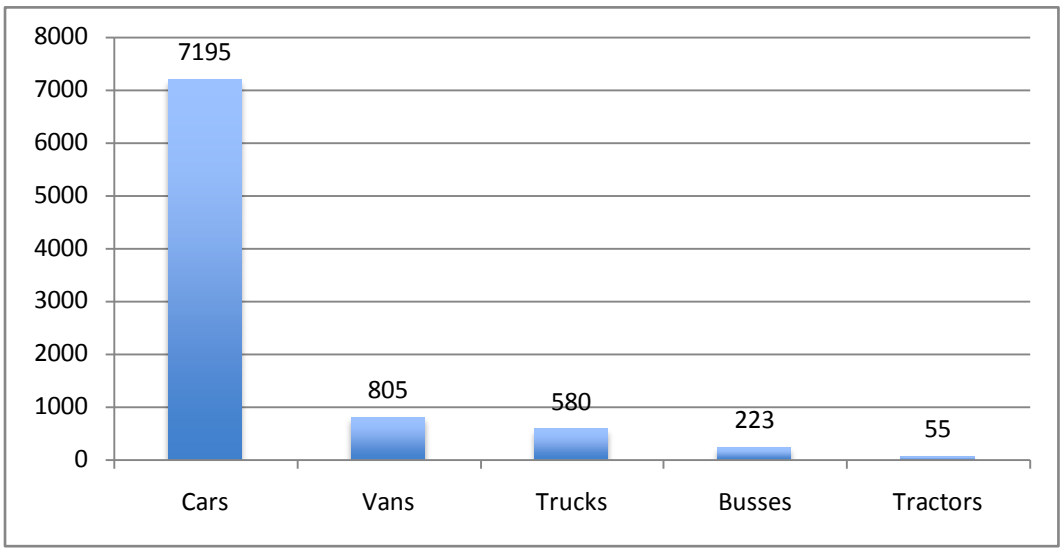


Figure 10: Vehicles involved in collisions with bicycles (2001-2008) [Statistik08a].

Furthermore, choosing cars will make testing substantially easier compared to choosing trucks simply from the logistical aspect of it: Finding an available truck and truck driver may prove troublesome.

2.3.1 CHOOSING ACCIDENT SITUATION

As mentioned the focus will be on cars colliding with bicycles. The next natural step is to look into which situations these two road users most frequently collide with each other. Statistics Denmark divides accidents between cars and cyclists into six different, simplified cases. A figure and explanation of each is show in Table 1.

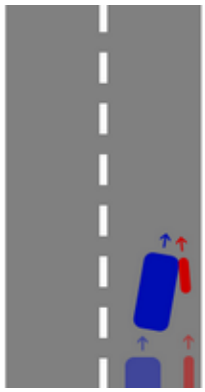
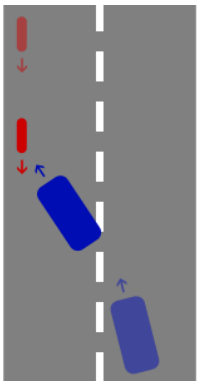
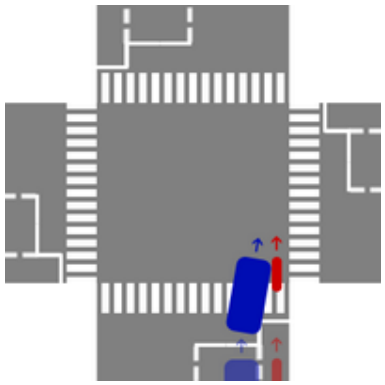
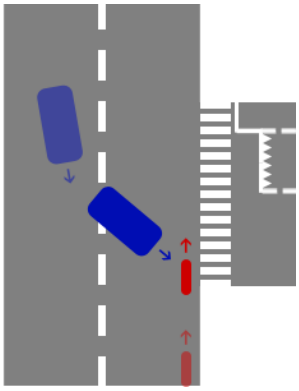
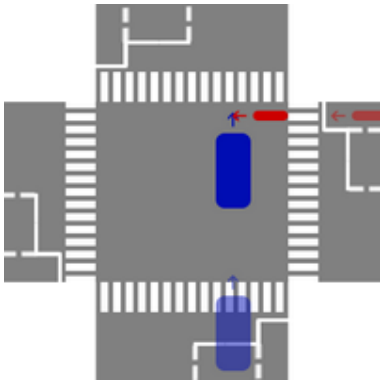
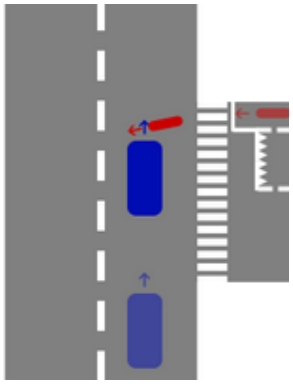
Case A  Vehicles on same road going in same direction without turning from road.	Case B  Vehicles on same road going in opposite direction without turning from road.	Case C  Vehicles on same road going in same direction, turning into T-junction, Y-junction, crossroads.
Case D  Vehicles on same road going in opposite direction, turning into T-junction, Y-junction, crossroads.	Case E  Vehicles on different roads meeting in crossroads without turning.	Case F  Vehicles on different roads meeting turning into T-junction, Y-junction, crossroads.

Table 1: Situations (cases) for collisions ending with injuries between cars and cyclists (2001-2008) [Statistik08a]

It is assumed that a single solution covering all cases would be unrealistic within the timeframe of this project, so one of the cases or a combination of similar cases has to be chosen.

The statistics show (see Figure 11) that Case F which is Vehicles on different roads meeting in T-junction, Y-junction or crossroads when turning - is consistently the case where most cyclist are injured. Closely following

Case F in the period between 2001 till 2008 is Case E where Vehicles on different roads meeting in crossroads without turning.

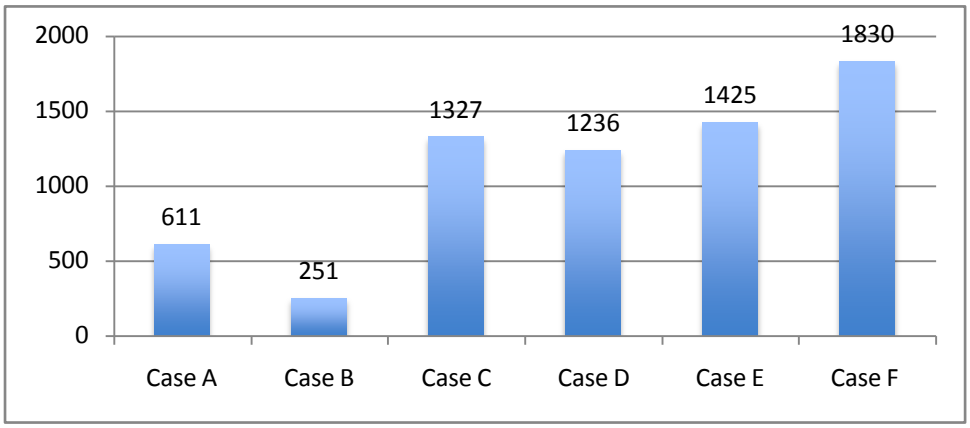


Figure 11: Injuries in collisions between cars and bicycles (2001-2008) [Statistik08a].

Following Case E is Case C where vehicles on the same road travelling in the same direction collide when one vehicle makes a turn - this can only be interpreted as being right turns. Based on statistics alone it would make sense to choose the case F situations, but this should not necessarily be the deciding factor. While any of all the six cases would have to include a function that warns the driver or applies the brakes before impact, the right turn situation differs by having the car driving alongside the cyclist before impact. Having the two parties travel alongside each other seems to offer some interesting detection opportunities, which in the end is the deciding factor: Right turn situations are chosen for this project.

2.4 SUMMARY

We have now reached a point where all the undecided factors mentioned in "2.3 Collision Statistics" have been clarified. The amount of people killed or injured while driving in a car is on a steady decline likely contributed by the advances made in active and passive safety. The cars are well-covered with a wide range of technologies that both helps prevent accidents and reduce injury in case of an accident. In short: It appears that the car industry is doing a good job at protecting its customers, and government initiatives such as the Danish Road Safety and Transport Agency certainly also plays its part.

This leaves us with the choice of VRUs, and statistics show that the fatality rate of cyclists are not on a decline potentially because of the lack of advances and contribution towards making the roads a safer place for them.

Statistics also support a need for reducing accidents occurring when a car collides with a cyclist in right turn situations.

The right turn situation where both vehicles travel in the same direction seems to offer a wider variety of possible solutions in compared to the other cases. Because of this we choose to base the project on right turn situations.

To sum up: The focus of this report will be right turn situations and the collisions they lead to between cars and cyclists, which leads to the final problem statement:

How can one prevent collisions between cars and cyclists
in right turn situations by the use of computer vision?

2.5 DELIMITATION

The final problem statement is focused around preventing collisions. There are two overall parts that need to be taken care of in order to prevent a collision though:

- A detection part
- A response part

The detection part is about detecting if a collision is potentially going to happen if none of the two parties involved change their path or velocity. The second part concerning response comes into play in the case where a detection has positively confirmed that a collision is likely to happen if no other action is taken. This response can either take place as a warning to one of the involved parties or as some kind of automated system taking over the control of the situation. In the case of a warning one or both of the involved parties have to be notified to change their velocity and path and prevent the collision.

The part that will be focused on in this project will be the detection part and being able to deliver a system output to be used by the response part when required. There will be no focus on how to develop any kind of automated braking systems and nor will there be any analysis on how to effectively warn the involved parties from a human-computer interaction perspective.

This implies that the system built in this report is not able to prevent the collisions without the remaining warning/response system. It is however presumed that the detection part of the system is the most crucial part, leaving the response system as the more straight forward part to develop.

In this report there is an equal sign between the task to detect a potential collision and to be able to prevent it. This might be a simplification, but it will be foolish to deny the relationship between the task of detecting a dangerous situation and preventing it. In all of the initiatives to prevent different kind of accidents mentioned in the state of the art chapter "2.2.1 State of the Art" the first part of the systems is to detect the specific situation, the next to react on it. In this project it will not be tested if the simplification is wrong.

2.6 SUCCESS CRITERIA

The goal of the project is first and foremost to answer the final problem statement. This however is something that potentially can be answered by providing a list of existing systems. For this project the goal is to develop a working prototype. In addition to this, a specific set of success criteria have been set up in order to more closely be able to determine the success of the project.

While other factors such as price and durability are important for a real-world product, it is the detection that will be the core of the product for this project and therefore detection effectiveness is the most crucial parameter when evaluating the solution. When considering detection effectiveness, one also needs to consider both the ability to detect True Positives (detection of an actual cyclist) and also the ability to not generate too many False Positives (detection of non-existing cyclist).

During the pre-analysis it was found that the currently most effective systems on the market can detect a potential collision in 9 out of 10 cases and give less than 1 out of 10 False Positives as mentioned in "2.2.1.2 Collisions Between Vehicles and VRUs".

We therefore set up the following criteria:

- A test in a real life situation which proves that at least nine out of ten potential collisions are detected.
- A test in a real life situation which proves that less than one out of ten positives are False Positives.

If this criteria is fulfilled it can then be concluded that the project is a success, because the final problem statement has not just been answered, but answered upon with a satisfactory answer. A discussion of how well the solution has been executed outside the scope of the success criteria will be done in the discussion chapter *7 Discussion*.

The final test will be performed by having one car and one cyclist ride alongside each other in a scenario as closely matching real life as possible. Exactly what that defines a realistic scenario will be found during the analysis. The involved parties will be the car and the cyclist respectively.

Another VRU such as a pedestrian is a case which it would be acceptable to detect, even though the core goal of this system is cyclists. This means that such a case will not be considered a True Positive nor False Positive.

If the results of the final test fulfil the two above criteria then the final problem statement is answered successfully upon.

2.7 METHODOLOGY

The following chapter will describe the scientific approach to this rapport. Medialogy is a natural scientific and technical education. This means that a technical and scientific approach is needed in order to solve Medialogy related problems. In the following project, different computer vision techniques will be analyzed and discussed in order to find the best possible technical solution for this projects problem statement. There will however also be some humanistic aspects of this project such as analysis on vehicle drivers' and cyclist' behavior in the traffic.

2.7.1 METHODS

This chapter will first summarize the set of methods that is used throughout the report, and secondly but most importantly describe why this set of methods is chosen. The methods of the report are described in order to ensure the validity of the results and the possibility to recreate the same results obtained in the project.

Research and Literature

In the innovative field of road safety new products and initiatives based on computer vision are rapidly generated. In order to keep this report up-to-date, scientific reports and journals are preferred opposite to books which content have a tendency to be overtaking by the reality before they hits the bookshop. Thus peer-reviewed reports and journals makes up the scientific and technical foundation for this report.

However scientific books will be used where the topics of interest are less evolving. Due to natural differences in traffic culture and regulations one need to look primarily at Danish literature and sources for the results to be accurate for this project.

Technical analysis

In order to develop an actual prototype, different computer vision techniques must be examined so that the most appropriate ones with regards to the final problem statement can be chosen.

The research on computer vision techniques will be done partly by analysis but also by testing different image processing techniques in order to get a better overview on the options the developers have for programming the final product.

Software requirement specifications are going to be defined and described using use cases before the design and development part. Use cases will provide the designers with scenarios of how the user will interact with the system which will provide an overview of what needs to be designed.

Development strategy

The final design and implementation of the product is going to be done with the assistance of the development tool of SPU-UML (Structured Program Development - Unified Modeling Language) [Hansen05]. The W-model is chosen and will be used to support an iterative process. This ensures that the designers have a program up running already in the start of the process which should prevent the project from going too far out of pass that could potentially be a dead end. The program is then improved continuously and more features are added.

3. ANALYSIS

At this point the problem area is defined and the specific areas of this problem now needs to be analyzed in order to gather enough information to be able to create the requirement specification the analysis should end up in. This specification should serve as a guideline for the design which is to happen as the next step in the development. The requirements need to specify a product that can solve the problem in the final problem statement and at the same time the requirements should be technically realistic. In order to know the exact requirements for the product a number of areas will be researched.

When designing a product for a given problem one needs not just to know the main problem but also all the underlying aspects of the situation. In our case it is important to know at which places, at which speeds and at which angles the right turn collisions actually happen. This will help narrowing down the exact requirements for the technology. An in-depth analysis of the right turn accident is therefore the first part that needs to be researched.

A lot of different techniques for computer vision in regards to traffic safety were found in "2.2.1 State of the Art". It is still unclear exactly which ones that will suit this project. Limits of computer vision needs to be determined to figure out what will be doable. As a part of this a number of small-scale tests will be done on different technology aspects in order to determine the feasibility of these.

One of the important things which needs to be analysed is the exact place that the system needs to be located, because whether the final product is static (placed stationary) or dynamic (in motion) will make a significant difference. This will require an analysis of multiple feasible solutions and subsequently a choice will be made.

The analysis will end with a final requirement specification, which needs to build on all the sub-conclusions found in the just described parts of the analysis. As mentioned, the requirements need to specify a product that can solve the problem in the final problem statement and at the same time the requirements should be technically realistic.

Based on the above structure the analysis will start by an investigation of where and how the right turn accidents occur.

3.1 ANALYSIS OF RIGHT TURN ACCIDENTS

In order to develop a product that helps with avoiding collisions in right turn situations, it is crucial to know all information related to this special kind of collision. This chapter will answer five questions. To know what the product must do in order to solve the problem (prevent collisions) one needs to know why the collisions happen in the first place (1). To know where the product should be placed physically it is important to answer the question of where the collisions happen (2). Because of the nature of the problem all collisions will happen outside in the traffic as opposed to a easily controlled environment inside a building. This means that the time of the collision during the day must be determined to know the light conditions (3). If a collision is likely to happen it is important to know where to look in order to prevent it. That is why it is important to know how and where the involved parties are placed just before and during the collision (4). Lastly it is important to know the speed of the car and the cyclist during the right turn accident (5).

The information gathered in this chapter is primarily based on research done by the Investigation Team For Road Accidents (HVU) which is an organization under the Danish Directorate of Roads. HVU has the purpose of collecting information about traffic collisions which will be used to improve traffic safety. The research about right-turn accidents have been conducted in the spring of 2005. HVU gathered data in a period of 8 months in

cooperation with all Danish police departments. The final rapport by HVU contains an in-depth analysis of 25 collisions between a right turning truck and a cyclist [Havarikommissionen06]. Unfortunately this was the only very detailed report about right turn accidents in Denmark. Since this report is about trucks, not cars, other reports, for example about similar accidents, are used to substantiate the results. One of those is [HavariKommissionen08], another report from HVU, analyzing 30 crossing accidents between cars and cyclists in non-regulated intersections.

It has previously been determined that a system for this project should prevent right turning cars colliding with a cyclist. The reason why HVU is interested in right turn collisions including only trucks, is that a fatal outcome is more frequent when a truck is involved, see Figure 12. That is also the reason why no similar investigation regarding collisions between smaller vehicles and cyclists has been made for the Danish traffic situation, as claimed by the chairman of HVU [CDa]. It is interesting to note that the number of fatal outcomes is higher for trucks even though figure Figure 10 found that by far most right turn accidents involved cars.

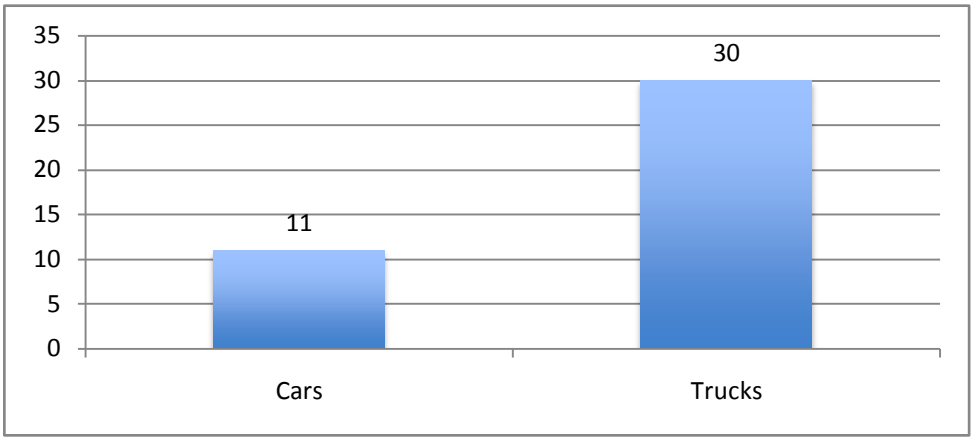


Figure 12: Right turn accidents with fatal outcomes
between cyclists and cars, as well as cyclists and trucks (2003-2008) [Statistik08a].

The analysis of the 25 right turn accidents from [HavariKommissionen06] is valuable because of their level of detail, but the limited number of accidents mean that no strong statistical result can be proven.

3.1.1 WHY THE COLLISIONS HAPPEN

As described by HVU and analyzed in the following section the reason why the 25 accidents happens, can be divided into three main reasons. These reasons are related to the driver of the trucks, the surrounding traffic in which the accidents happened, and the conditions of the vehicles which includes mirrors and such.

The Driver

Not surveying the situation is one of the most frequent factors which results in collisions since it happened in all 25 collisions described by [HavariKommissionen06]. According to the report it has been concluded that all of the truck drivers had the opportunity to see the cyclist at some point before the collision e.g. via the mirrors if they had been adjusted properly.

Taking this into consideration one must still not forget that driving and making a right turn with a truck is a more complex maneuver than turning with a car. However, [HavariKommissionen08] also conclude that not surveying the situation before turning was also the main reason for the accidents in that report (it was a factor in 15 out of 30 accidents). Even if we did not have these results, it seems like a logical conclusion to determine that the accidents happened because the driver did not pay attention, since he has the duty to give the right of way.

Not surveying the situation by the cyclist has not been mentioned as one of the reasons resulting in the collision by the [HavariKommissionen06]. This is because the cyclists have the right of way, and the truck drivers have the responsibility of correctly getting an overview of the situation before making the turn.

Other factors such as fatigue, bad physical conditions or mental conditions such as family issues, and a tight work schedule could also have played a part in the parts lack of attention towards each other.

Surrounding Traffic and the Design of the Road

Both [HavariKommissionen06] and [HavariKommissionen08] also mention the surrounding traffic as a reason for why the vehicles involved in the collisions did not pay enough attention to the cyclist and therefore ended in an accident. The drivers had drawn their attention toward another car, the lane of the road or some construction work, for example. More natural factors such as a bright sun blinding either the driver or cyclist had also played a part in some of the collisions.

The Trucks

21 out of the 25 trucks involved had not adjusted their mirrors correctly. This means that even though they may have oriented themselves via the mirrors, they would most likely not have had the opportunity of spotting the bicycle because of the wrongly adjusted mirrors. But this cannot be generalized so that the information can be used with cars. Trucks have a large number of mirrors compared to cars which will make it harder to adjust all of them.

Summary

Overall it can be assumed that if a car driver spots a nearby cyclist coming up on the car's right side, then the driver will not make the turn. This means that the problem behind right turn collisions happening must be that the driver has either not seen the nearby cyclist, since it is always the driver who has the duty to give way, or thinks he has enough time to make the turn. This also seems to be substantiated by the statistics mentioned in this chapter.

This means that one should look make a solution that cover both of these problems, so the system will function regardless of which of these is the problem in a certain situation. More specifically, the system should be able to detect the cyclist and it should be able to detect the cyclist within a time frame where the car has time to brake.

3.1.2 WHERE THE COLLISIONS HAPPEN

This section strives to determine the outer circumstances in right-turn accidents between a car and a cyclist. In this case outer circumstances mean everything related to the collision except matters related directly to the properties of the involved parties and their behavior before and under the collision. Questions like where the right turn accidents occured will be answered. The following section will still be based on the HVU reports about right turning trucks, [Havarikommissionen06] and [HavariKommissionen08], about crossing accidents between cars and cyclists.

Knowing where the accidents happen is important when choosing where to mount a static system, for example.

Location of the collisions

Of the 25 right turn accidents analyzed in [Havarikommissionen06] 23 occurred in urban areas. Even though not directly relatable, since the report is not only about right turn accidents, [Havarikommissionen08] also states that 24 out of the 30 their analyzed accidents happened in urban areas. Out of those 23 in

[Havarikommissionen06] the majority (15) happened in urban areas with housing and the rest in industrial environment and ribbon development.

In general the crossroads with the accidents are relatively busy. The degree of activity is measured as the daily traffic rate measured over the span of a year. It tells the number of vehicles that pass through a crossroad in 24 hours as a mean over one year. The least busy crossroad has a rate of 9,000 and the rest were equally distributed over the interval 9,000-26,000, one being 41,000. This gives an indication of where a system could be placed, if it for example is chosen to be stationary.

The conditions of the road and its surroundings

It is hard to find a tendency towards a specific kind of condition of the road where the right turn accident happens. In [Havarikommissionen06] 14 of 21 accidents occurred in crossroads with bicycle paths and the remainder in crossroads without a path. When the road has a bicycle path the path sometimes merge with the turning lane just before the crossroad. This is done so the cyclist and motorist are forced to take notice of each other before crossing the intersection. One accident out of 21 from [Havarikommissionen06] occurred under such circumstances. We were not able to find data about the frequency of bicycle paths on Danish roads to give a qualified estimate on which road type is more dangerous.

3.1.3 WHEN THE COLLISIONS HAPPEN

Due to time constraints, it will not necessarily be a priority to create a solution that works equally well in all lighting conditions. Because of this it is also relevant to look into when the accidents happen distributed on 24 hours of the day. Based on this analysis specific lighting conditions will be chosen for the system.

All 25 accidents from [Havarikommissionen06] happened from 6 a.m. till 6 p.m. This is also substantiated by [Havarikommissionen08], an American paper, [Turner-Fairbank97], and a German paper [Niewoehner05] which made an in-depth analysis of 90 accidents involving a right turning truck and a cyclist. They also tested throughout 24 hours and found that all accidents happened from 6 a.m. till 6 p.m. Although German or American traffic properties should not be directly compared to Danish circumstances it is presumed they substantiate the tendencies.

3.1.4 WHERE THE CYCLIST IS PLACED COMPARED TO THE VEHICLE

In order to develop a system that can help reduce the amount of right turn collisions, information about the bicycles' placement relative to the vehicles are needed. More specifically, it is important to know whether or not the car is in front of the cyclist when the collision occurs or if the cyclist is at the front end of the car when the collision occurs. Knowing this will increase the chance of finding the right setup for detecting the system so that it will know where to look. Data gathered in 2006 by HVU will again be used [Havarikommissionen06]. For all of these results it should be stressed that truck drivers have a very limited view area around the truck compared to a car, and as such it is unknown whether the situations are the same for right turn accidents with cars. Since no Danish report about right-turn accidents with cars and cyclists has been found, the search was widened to foreign reports, and, as mentioned, a very extensive American report from 1997 was found [Turner-Fairbank97], dealing with 113 right-turn collisions where the cyclist and the car was driving in the same direction. The circumstances of those collisions seem similar to the ones from [Havarikommissionen06], since most of them happened during the day and in urban areas. Therefore these results will also be mentioned in the following sections.

Cyclist in front of the truck

In 11 (44%) of the incidents from [HavariKommissionen06] the bicycles were placed in front of the trucks before the collisions, for example by having overtaken the truck or by starting to drive first after being stationary in an intersection. Those cyclists were then hit from behind by either the front or the right front corner of the truck (one of them was hit by the left side of the truck, because he atypically turned left after driving past the truck). In [Turner-Fairbank97] 84 (74%) of the collisions, the car was overtaking the cyclist before the accident, i.e. the cyclist were in front of the car before the collision.

Cyclist next to the truck

In 3 (12%) out of the 25 right turn accidents from [HavariKommissionen06] the cyclist arrived on the right side of the truck just before the truck started to turn. This means that they both entered the intersection at almost the same time and it ended up in a collision where the bicycle was hit by the front right side or the right side of the truck. No results from [Turner-Fairbank97] state that the cyclists and cars were almost next to each other.

Cyclist behind the truck

In the final 11 collisions described by [Havarikommissionen06] the trucks completed their right turn while being in front of the cyclists. This resulted in a crash where 5 of the bicycles got hit by the right front side of the truck and 6 got hit by the right side of the truck. In [Turner-Fairbank97], 12 (11%) of the cases had the cyclist overtaking the car (i.e. the car was in front of the cyclist).

The situation of the rest of the cases from [Turner-Fairbank97] are undetermined. [Turner-Fairbank97] gives a significantly different results than those from [HavariKommissionen06], i.e. in [HavariKommissionen06] 11 accidents happened when the cyclist was in front of the car before the accident and 11 happened when the car was in front of the cyclist before the accident. In [Turner-Fairbank97] these numbers were 84 and 12 accidents respectively. This could both be because other factors are relevant in the American cases, or simply because only 25 Danish accidents do not give a representative view of right-turn accidents in general.

Regardless of whether or not the Danish or American data are more correct, it is obvious that situations where a car is in front, behind or next to the cyclist before the accident are frequent enough to warrant the creation of a system that can prevent collisions in all three circumstances.

3.1.5 SPEED AND REACTION TIME

In order to estimate how long reaction time a future system will have, data about the speed of cars making a right turn and the cyclists are needed. Since data concerning the speed of cars during right turns are not described in any of the reports found, an observation test will be conducted to collect the needed data. The observations will take place in the field, meaning that the cars will be observed making right turns in the real world, and data will be collected to calculate the speed of the cars when turning.

Data was collected using a Canon XL2 video camera which filmed cars making right turns. Two crossroads were chosen, one with a speed limit of 50 km/h and one with 70 km/h speed limit, partly chosen based on [Turner-Fairbank97], where circa 62% of the right-turn accidents happened on roads with speed limits of 50-60 km/h. There was no right turn traffic lighting in any of the two intersections. Distance markers were placed so the speed of the cars could be calculated using the length between the markers and the amount of time it took to complete the turn.

Figure 13: One car at the first marker in the first frame
and at the second marker in the second frame.

All cars turned with a speed below 35 km/h. The distribution of the driven speeds is shown in Table 2.

The speed of personal vehicles [km/h]					
Speed	0-10	11-20	21-30	31-40	**Total**
Number of cars	6	33	30	2	71

Table 2: Distribution of cars' speed when turning.

The observed cars had a tendency of slowing down when approaching the turn. Only two cars turning with a speed above 30 km/h. 6 of the cars nearly stopped fully when turning. Combined all of the cars had an average speed of 18.7 km/h.

Data about the speed of the cyclists is provided by the HVU report. The cyclists had a speed of 0-45 km/h just before the collisions. The distribution of the speed is shown in Table 3.

The speed of the cyclist [km/h]	Total
0 - 4,9	2
5 - 9,9	1
10 - 14,9	8
15 - 19,9	5
20 - 24,9	4
25 - 29,9	0
30 - 34,9	1
35 - 39,9	1
40 - 45	3
Total	25

Table 3: Distribution of cyclists' speed during right turn collisions [Havarikommissionen06]

It is presumed that the speed of the cyclist will not significantly change in right turn situations with cars.

3.1.6 CONCLUSION

The majority of this chapter was divided into five sections; why, where and when accidents happen, where cyclists are compared to the vehicles during and just before the accidents and the speed of the parties involved. A prominent reason of why the accidents happen seems to be a lack of overview on the driver's part, which means they either do not see the cyclist or misjudge the distance to the cyclist. The final product of this project must take both of these factors into consideration.

Most of the collisions occured in urban areas in intersections with a traffic rate of minimum 9,000, this is useful for selecting a place where the system should be, if the system is chosen to be stationary, for example. This might also help to set some parameters that the system must be able to fulfill as a minimum. No data could be found concerning the frequency of bike paths and how this impacts the accidents, so the system should work regardless of if the cyclist is on a bike path or not.

Almost every right turn accident happen in daylight so the solution should be implemented with that in mind as well. Most accidents seem to happen when the vehicle is overtaking the cyclist, but accidents also happen when cyclists are overtaking or next to the vehicles. Therefore the system should work in all three situations.

The speed of the involved parties were under 25 km/h for trucks immediately before the collision and up to 45 km/h for cyclists (none standing still), which sets a requirement for the level of performance of the parts used in the final product.

Before looking into the field of computer vision in detail, an overall classification of possible solutions will be made. This is done to get an overview of how to proceed with the analysis and development of the final product.

3.2 TWO SYSTEM CLASSIFICATION

Regarding the physical placement of the system two variations seem to cover all the possible scenarios: a stationary system, from here on referred to as the Static System, and a system in motion, from here on referred to as the Dynamic System. An example of a static system would be where the camera is mounted somewhere in an intersection, and the dynamic solution would be mounted on one of the involved parties, i.e. the car or the cyclist.

In chapter "3.3 Computer Vision Techniques" specific techniques that seem useful for one of the systems will be analyzed. This is done to be able to choose one of the systems on a well-researched foundation. However it might turn out that a clear answer to some of these questions will not be easily obtainable, in that case those will be identified as subjects that require testing later on.

At this state some differences between these systems can be noted; a dynamic system, placed on the vehicle, would only need to detect the cyclist and his position in relation to the car. A static system would need to be able to detect the cyclist and the car, and then calculate the distances between these. Relevant techniques for each system will be analyzed and discussed in the following chapter.

3.3 COMPUTER VISION TECHNIQUES

It is now relevant to take a more in-depth look at relevant computer vision techniques. Since the accidents happen because of a lack of overview, as stated in "3.1.1 Why the Collisions Happen", it is necessary to aid the driver with this. This can be done in different ways, but according to the final problem statement it should concern computer vision techniques. However, according to "2.5 Delimitation", this system should not warn the driver, only detect the cyclist.

For each technique presented pros and cons related to the two possible systems will be listed to the possible extend with the current, limited amount of knowledge in these fields. This is done to be able to choose the techniques that will be used.

During this project references to the OpenCV library will be used. This is a library of programming functions for real time computer vision, originally developed by Intel. The built-in functions make use of assembler[2] optimization and Intel's SSE[3] optimizations and are thereby most likely faster than a custom implementation. [Bradski08].

Before looking at specific computer vision techniques, some introductory knowledge of the field must be acquired.

Digital Images

To better understand the data, i.e. digital images, which the following computer vision techniques are working with, more information is needed. A digital image is represented in the form of a two-dimensional array of pixels. This array consists of data for each pixel in the image. Some of the most popular data formats are called YUV, HSV and RGB of which RGB is one of the most common for digital images. RGB is an additive colour system, and RGB stands for Red, Green and Blue. RGB is the primary colours in such digital images. The three colours mixed together gives white at full values and all other colours depending on various combinations.

YUV colour model is used for analogue televisions and many digital compression methods. HSV is short for hue, saturation and value. Hue is the colours chosen while increasing and decreasing the whiteness corresponds to

[2] Low-level language allowing for highly efficient code.

[3] A specific set of high-performance registers optimized for multimedia calculations.

adjusting the saturation and increasing the black colour in the image corresponds to lowering the intensity [Moeslund08].

Histogram

A histogram is used to visualize the distribution of pixel values in an image and is built up like a coordinate system. The x-axis denote pixel values, ranging from 0-255 (black to white) in an 8-bit image, and the y-axis denote the amount of pixels with each pixel value. The histogram is useful for many things, e.g. when thresholding, which will be explained in "3.3.3.1 Point Processing".

At this point it is not known which techniques that fit best to the Static System or the Dynamic System so a broad selection of techniques which seem useful and relevant (e.g. based on "2.2.1.2 Collisions Between Vehicles and VRUs") will be listed and explained. This chapter and the techniques are divided into the following categories based on a framework created in [Moeslund08]:

- Image Acquisition
- Pre-processing
- Segmentation
- Representation
- Recognition

3.3.1 IMAGE ACQUISITION

Image Acquisition has to do with everything concerning the setup of the camera, e.g. camera type, camera settings, optics, light sources, etc.

Stabilization

The first thing to look at is the input source, this is often a camera or a webcam which delivers a series of images also called frames. Image stabilization is a technique used for both reducing blurriness in still images and for video streams to reduce the amount of frame-to-frame jitter, also called shakiness. Image stabilization is something that is useful for other image processing and tracking techniques as those e.g. work better when the backgrounds are as similar as possible, as this makes the moving objects more distinguishable. There are two main methods to stabilize an image: Optical stabilization and digital stabilization [Anderson99].

Optical image stabilization is a technique implemented in the lens itself where the lens is moved according to input provided using angular velocity sensors (gyroscopes). Digital image stabilization takes another approach which does not involve physical movement. Instead a border buffer surrounding the visible frame of the image is used to minimize the amount of changes in the image. This is done by making use of the buffers on each side of the image depending on camera motion. So if the camera moves to the left, then some of the right side buffer will be used for a little while to create a smooth right to left transition. Another variant of this is without a buffer where the image instead is cropped slightly to achieve the same result. This digital stabilization usually happens in the camera and is then called electronic stabilization.

Another digital approach is by doing it a post-process by tracking features in an image and then trying to keep these features at consistent positions. Mid-end cameras and up usually have image stabilization, while very few web cameras have this feature. This means that one can end up having to implement the image stabilization in the software.

Stabilization does not seem to be very useful for the Static System, since stabilizing an image from a static camera is useless if it is mounted properly. For the Dynamic System a camera with stabilization would be

preferred, since a stabilization implemented in the software would increase the computational time of the final system, and no matter how small this is it would be preferable to do as much as possible in-camera.

3.3.2 PRE-PROCESSING

Pre-processing covers the process of doing something with the image before the actual processing begins, e.g. converting the image to grayscale or cropping it.

Region of Interest

In image processing a large image (i.e. many pixels) is not always a benefit. The reason for this is, that calculations are often done on a pixel by pixel basis, and an increasing amount of pixels makes these calculations more demanding. To solve this, it is possible to use a Region of Interest (ROI), which is simply a region in the image that defines the pixels of interest. All other pixels are ignored [Moeslund08].

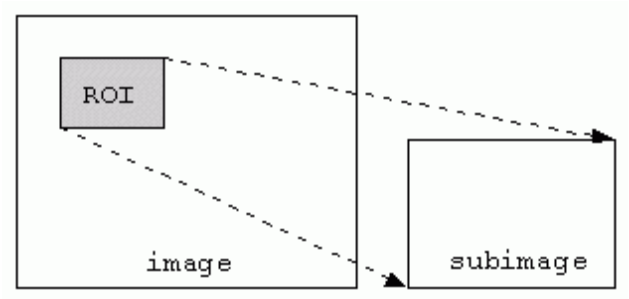


Figure 14: Region of Interest (Nash Ruddin) [NashRuddin09].

3.3.3 SEGMENTATION

Segmentation is the process of partitioning a digital image into several segments to simplify and make the image easier to process or analyze. This chapter will focus on some of the simple and relevant segmentation techniques.

3.3.3.1 POINT PROCESSING

Point Processing concerns techniques that change a pixel only based on the corresponding pixel of the original image.

Thresholding

Thresholding is a technique used to do point processing operations on an image. E.g. if you do thresholding on an 8-bit grayscale image all pixels with the value under a certain threshold, e.g. 127 and below is set to zero while all values above 127 are then set to 255. This produces a binary image, which means that all pixels are either white or black. A lot of information is lost with thresholding, but when doing image processing minute details in the image is often more of a hindrance to what one is trying to achieve. Thresholding is often a key step to segmenting the foreground (information) from the background (noise). The threshold can be decided depending on the image's histogram and what one wants to isolate. An ideal histogram has two peaks in each side, with one peaking containing the information for the foreground pixels and the other containing the information for the background pixels, making it easy to set the threshold in between [Moeslund08].

Since thresholding is often done before using other segmentation techniques, for example related to noise reduction (explained in "3.3.3.2 Neighborhood Processing"), it is likely that it would be used in both final systems.

3.3.3.2 NEIGHBORHOOD PROCESSING

Neighborhood Processing concerns changing a pixel in an image based on the neighboring pixel values in the original image.

Mean and Median

Noise often takes the appearance of single pixels that stand out from their neighbor pixels and thereby create visual artefacts. Noise can be many things though. The version where pixels have a value of 0 or 255 is called salt-and-pepper noise. Noise of a less extreme kind also exists, this aren't as clear artefacts but is also harder to remove.

A simple way to remove noise is to use the mean (arithmetic average) filter which uses neighborhood processing to replace the noise pixels with the average value of the surrounding pixels. Usually it is important to only take the surrounding pixels and not include the middle pixel (that means a total of 8 pixels of a 3x3 filter). This filter can vary in size, depending on how extensive the filter should be.

A better result is often yielded by using a median filter instead of a mean filter. A median filter works by taking the median number of all the neighboring pixels [Moeslund08]. A comparison between the mean and median filter can be seen in Figure 15.

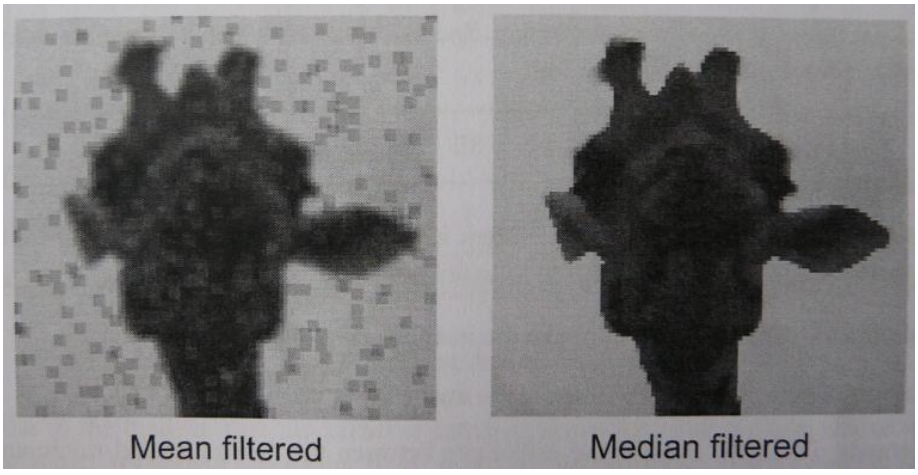


Figure 15: Comparison of mean and median filter [Moeslund08].

If necessary the mean and median filters will be used to remove noise for the final systems.

Correlation

Correlation covers techniques using so-called kernels to apply changes to every pixel in an image. According to [Moeslund08] the mean and median filters are not included in Correlation, since the Correlation kernels play a more active role.

Dilation and Erosion

Dilation and Erosion are two methods which can be used to clean up noisy images instead of using mean and median filters. They are effective at removing more than just isolated noise but also just to get rid of small features of an image. If the final system is to try and find objects with the help of thresholding then some amount of dilation and erosion probably has to be used.

Erosion and Dilation use what is called Fit and Hit methods, respectively. For example, a Hit method overlays a kernel on an image (for example a 3x3 kernel with pixels of value 1) and if just one of the pixels on the image, which are lying inside this kernel, is 1, we have a "Hit" and the center pixel's value is changed to 1. Similarly,

with the Fit method, all underlying pixels must be 1, with the same 3x3 kernel, if not, the center pixel is changed to 0. Erosion and Dilation is simply applying Fit and Hit methods to an entire image. It also possible to combine Dilation and Erosion, this is called Compound Operations, and yields different results. If Dilation is done and then Erosion, it is called Closing, this can be used to close holes in BLOBs and then remove noise. If you run Erosion first and then Dilation it is called Opening. It is also possible to combine Closing and Opening, for example this is useful if we have holes inside a main object *and* small noise objects [Moeslund08].

Template Matching

Template matching is used to find parts of an image which match a template image. The difference from silhouette matching, a similar technique discussed in "2.2.1.2 Collisions Between Vehicles and VRUs", is that template matching takes into account the surface appearance whereas silhouette matching obviously only looks at the objects' contours. Template matching slides the template from the top left to the bottom right of the image, and compares for the best match with the template. The result is a guess on where the template image is matching the source image best [Moeslund08].

Since this technique specifically tries to find a template in an image, it seems like a viable choice for one or both of the systems.

Sharpen and edge detection

Sharpening is a technique which can increase the acutance[4] in an image. This is often achieved by applying an unsharpen mask, which is a process where a copy is made of the image and blurred with e.g. Gaussian blur[5] and then subtracted from the original image based on a threshold. When used specifically with Gaussian blur the technique is called Gaussian sharpen.

Edge detection is a technique used to find edges in an image, edges being defined as pixel positions with a significant change in gray-level values, and can be considered a special case of sharpening. With correlation a horizontal and vertical kernel filter is run through the entire image and the filters are then applied to each pixel and combined. The most well known kernels are the Prewitt kernels and the Sobel kernels.

The pixels with a significantly different value compared to the surrounding pixels are multiplied with a higher integer to increase the difference and thereby highlight the edge.

By looking at the image through each row and column the pixels with a suddenly high value change compared to the previous set of pixels are set to a higher value and values where the change in value are not significant is set to a lower value. This gives an image where all edges are labelled with a high pixel value and the areas with no changes in intensity are set to a low pixel value, highlighting the edges.

Edge detection may well come into play during this project when detecting and tracking the involved parties as it is often used in combination with other techniques such as corner detection.

3.3.4 BLOB ANALYSIS

BLOB stands for Binary Large OBject, the term "Large" indicates that only objects of a certain size are interesting and "small" objects are usually noise.

[4] The amplitude of the derivate of brightness.

[5] Blurring an image by the use of a Gaussian function.

According to [Moeslund08] a BLOB Analysis section sometimes replaces the Representation and Recognition chapters, depending on the project at hand. This is also the case for this project, since the techniques in this chapter seem sufficient to represent and recognize the BLOBs in either system.

BLOB Analysis consists of two parts:

- BLOB Extraction, i.e. separating objects in a binary image.
- BLOB Classification, i.e. indicate which objects we are looking for.

3.3.4.1 REPRESENTATION (BLOB EXTRACTION)

The purpose here is to isolate the BLOBs, and a BLOB is, as mentioned, a group of connected pixels. Whether or not pixels are connected is defined by their *connectivity*, so pixels do not have to be exactly next to each other to be defined as connected (e.g. it can be chosen that BLOBs two pixels apart are still denoted as being connected). Different kinds of algorithms can be used to find BLOBs, one of these is the Grass-fire algorithm.

Grass-fire Algorithm

The Grass-fire algorithm is able to separate individual BLOBs on an image and give them index numbers (BLOB ids). It does this by going through the image and then indexing each blob, e.g. by applying a variant of flood fill[6] for each BLOB.

The pseudo code for Grass-fire:

```
for all pixels
     check if BLOB pixel is found
     mark with BLOB id
            check all neighborhood pixels for BLOB pixel
```

Code Block 1: Pseudo code for the Grass-fire algorithm

The Grass-fire algorithm might be useful if it is necessary to distinguish between several BLOBs in the final systems [Moeslund08].

3.3.4.2 RECOGNITION (BLOB CLASSIFICATION)

After extracting the BLOBs it is necessary to classify them. For example, if we have 3 circles and a square, and we want to find the circles, we need to classify all BLOBs as either a circle or a square. This process consists of two steps, first each BLOB must be represented with a number of characteristics, denoted as *features*, and then a matching method is applied to compare these features with the features of the object we are looking for.

Feature Extraction

Feature Extraction concerns extracting unique features from a BLOB, making it possible to identify and analyze this BLOB. Before doing that it is crucial to ignore all BLOBs connected to the borders of the image, since it is necessary to see the entire BLOB before it can be identified. A number of features can be extracted from each BLOB and the following is a description of some of the common features.

Area is the number of pixels a BLOB consists of and can be used to remove BLOBs with a certain size, *Bounding box* is the minimum rectangle containing the BLOB, this can be used as a ROI, *Center of mass* is the location on

[6] Coloring a flat shaded area.

the object where you would place a joint to balance it, *Perimeter* is the length of a BLOB's contour and *Circularity* defines how circular a BLOB is, which obviously can be used to find circles, amongst other things [Moeslund08].

Feature Matching

When the BLOBs have been identified and features extracted, it is possible to start a feature matching. To do this we create a *prototype model* of the object we are looking for. For example, if we are looking for a circle we will look for similarity to a circle prototype among the extracted features. It should be noted, that it is necessary to expect some deviations from a perfect circle, so deviations must be implemented as well. In the case of a circle, one can say it must have a circularity of 1 (perfect circle) with a deviation of +- 0.15. This is always a balance that must be found individually for each project or situation [Moeslund08].

Looking at the abilities of feature extraction and matching it appears to be a technique which can be useful for both systems, as long as a BLOB with features that do not change radically, exists.

3.3.5 OTHER TECHNIQUES

In "2.2.1.2 Collisions Between Vehicles and VRUs" a lot of the projects used stereo vision and the Kalman filter, these specific methods are described in this section. Optical flow, which is the estimation of motion, was also found when looking for techniques relevant to a dynamic system. Therefore this field will also be described in this section.

3.3.5.1 STEREO VISION

Stereo was a technique that was very commonly used in the projects found in the "2.2.1.2 Collisions Between Vehicles and VRUs". Binocular stereo or stereo vision is the process leading to the sensation of depth utilizing two cameras looking in the same direction, each with slightly different positions and angles. This is very similar to the human visual system, where a main reason for why the human have two eyes is to better be able to perceive depth [Wolfe09]. Computers can also be used to measure depth and distance in this way and the process is then called computer stereo vision. This can e.g. be used to isolate all targets within a certain distance. To make stereo vision usable it is necessary to find similarities between two input sources in order to be able to identify one object in two images.

There are several types of stereo methods:

- Area-based stereo
- Feature-based stereo

The area-based solution is the simplest method, but it comes with a cost. Area-based stereo looks through two images to find common matches with the use of kernels, similar to Template Matching (explained in "3.3.3.2 Neighborhood Processing"). A kernel could be a 3x3 grid put on top of 9 pixels in the left camera's image. A similar kernel then looks for the same grid in the right camera's image. This method demands a lot of computational power and multiple matches can be found.

The featured-based method tries to find features with edge detection, i.e. it tries to find the object's features and concentrates on trying to match these. Because of this it is also faster than area-based stereo, but it also gives a sparse map of depth [Bradski08].

Disparity Maps

Disparity maps are gray scale images that present the depth perception, these are often used in conjunction with computer stereo vision. As seen in Figure 16, the lighter areas of the disparity maps are closer.

Figure 16: Disparity map (Programmers United Develop Net).

Stereo vision seems very useful, but many papers stress that this technique has a high computational time. It is also obviously more time-consuming to set up and calibrate two cameras instead of one [Gandi08].

3.3.5.2 KALMAN FILTER

The Kalman filter can estimate the position, trajectory and speed of a moving object. Mentioned in "2.2.1.2 Collisions Between Vehicles and VRUs", Kalman filter is the most applied technique concerning tracking in the field of image processing. The Kalman filter can ignore noise and still retain useful information when tracking objects that may disappear for a while, for example when other objects pass in between the camera and the tracked object [Welch06].

3.3.5.3 OPTICAL FLOW

Optical flow is the pattern of apparent moving objects, surfaces and edges in a visual scene. Optical flow techniques cover a variety of methods, e.g. motion detection, object segmentation or time-to-collision. It can also be used to compensate for movement in a moving video by comparing two video frames, for example, and that is where it seems to come in handy for this project. Using it like this makes it possible to do a background subtraction after compensating for movement, and thus see the differences in the two frames, i.e. spotting objects moving at a different velocity than the background.

Shi and Tomasi

One way to estimate optical flow is by finding the same feature in several images. Such a feature either needs to be a corner of have a texture that stands out in the images. An algorithm then needs to determine if these corner and texture features are good to track. In this case, the Shi and Tomasi corner detection algorithm can be used [Shi94]. The reason for mentioning it here is that it seems to be very popular and it is built into OpenCV.

Lucas-Kanade

The Lucas–Kanade Optical Flow Method [Kanade81] is one of most popular ways to estimate optical flow. The reason for choosing it here is that it is deemed to be the most reliable method to find local differences between two images in a scene [Lucas85]. It is also already built into OpenCV.

More specifically the algorithm can take the features found by the Shi and Tomasi algorithm and look for them in the next image frame and is then able to determine the displacement between objects and thereby their velocity fields.

Since there will probably be a lot of movement in a potential dynamic system, it is necessary to use methods or techniques that can function despite of this. Using optical flow seems like a viable solution, since it still works if

the platform (e.g. a car) the camera is placed on is moving. It will also work for a potential static system, but it seems like that system can be completed with less advanced techniques.

3.3.6 Relevant Techniques to be Used

A lot of techniques have now been looked at. However, nothing has been able to clearly show us what direction to take concerning choice of technique or system. Therefore an analysis test will be done on simple versions of the two systems. The purpose of this is mainly for us to gain more knowledge in computer vision fields related to our two systems. The purpose of the test is also to finally give an indication of which system we should focus on for the rest of the project, i.e. the Static System or the Dynamic System. Before the test it is necessary to choose some of the mentioned techniques to build the simple systems with. These techniques may not be the same techniques used for the final system. This section will try to argue what techniques that could be used for each of these simple versions.

A different set of criteria than the Success Criteria are chosen for this analysis test. The reason is that the purpose of this test is only to quickly gain some knowledge and to choose a system over the other. Basically, the systems should be able to successfully detect a cyclist and/or a car in a scenario similar to a real-life scenario. Because of this the focus for the chosen techniques will be on the simplest, most straightforward techniques and those with some basis in what we have already learned.

First, this section will divide the mentioned techniques into a dynamic and static part to finally determine the techniques or filters that will be used for this test (but, as mentioned, not necessarily for the final system).

The Simple Dynamic System

A dynamic system, meaning that the camera is mounted on something which is moving, must compensate for movement in some way. This can for example be done with stereo vision or by estimating optical flow. Optical flow is chosen because it seems more straightforward compared to stereo vision, where you need two cameras. Furthermore, a Medialogy project similar to this was created with stereo vision, and we would like to try a different approach [Zhang07]. More specifically the Lucas-Kanade method will be used, because this has already been researched, is very popular and seems to be a good staring point for calculating optical flow. Optical flow may also be sufficient to detect a cyclist to the right of the vehicle without other techniques, since it is not crucial to detect whether or not it is in fact a cyclist, only that it something which is moving alongside the vehicle, for this minor analysis test. This means that techniques like template matching or feature matching will not be necessary to verify if it is in fact a cyclist.

The Simple Static System

For The Simple Static System a lot of techniques can be used, but, as mentioned in the introduction of "3.3 Computer Vision Techniques", the most straightforward or simplest technique should be looked at. If this works there is no reason to look at more advanced techniques or filters, if it does not work more techniques or filters can be applied. The most straightforward method seems to be by doing a BLOB Analysis, and this may be enough to detect and/or track a cyclist and a vehicle with a static system. As such, the more advanced techniques and filters, such as stereo vision or the Kalman filter may be implemented if necessary. At this point it is not known which BLOB features will be used to identify the objects, i.e. the cyclist and the vehicle.

For both systems

The techniques mentioned in "3.3.1 Image Acquisition", which focus on optimizing the image, might be used for the test. However, because of time constraints the test will be as simple as possible, and if the preprocessing techniques seem to be necessary, they will be used.

3.3.7 CONCLUSION

There are many techniques that can be used within computer vision and most solutions will have to make use of a combination of several of these. Exactly which ones to use depend a lot on the situation and goal of the specific project. If the camera is located on a moving platform, such as a car, then image stabilization may be required, while a static system should not need this.

Only little information was found regarding performance (execution speed) issues for the techniques, but it is clear that it is something one has to be cautious of. Further testing of this probably has to be done in order to determine exactly how much of an issue it actually is.

A technique such as ROI can help speed issues, but is not something that will need to be dealt with from the beginning. If, however, the system runs into performance issues, then ROI is clearly one of the more effective techniques to help reduce this issue and should therefore be used in this case to ensure that the system at a speed matching the requirements.

Since thresholding is a fundamental part of several filters, i.e. edge detection, it is reasonable to believe thresholding will be used at some point. Similarly, noise reduction in the form of Mean and Median filters as well Erosion and Dilation will probably also be used. BLOB Analysis, more specifically Feature Matching will be tested to detect and track objects for The Simple Static System. If this is not sufficient other techniques will be applied. On the same note, Grass-fire may be used to extract the BLOBs. For The Simple Dynamic System, the focus is on implementing optical flow. Stereo vision seems harder to implement but it is also the best method for finding distances. However, this does not seem to be necessary, and optical flow will be used as a starting point. The Kalman filter seems unnecessary at this point.

At this point it is also clear that some hands-on testing of the techniques mentioned in this chapter is required in order to give a final answer to which ones to use and how. Exact computational requirements are unknown and will depend on specific implementation, so this will be one of the things that will need testing. The chosen techniques for the analysis test is, as mentioned, respectively optical flow and BLOB Analysis. If these techniques are not sufficient, more filters and techniques can be applied.

It is apparent that several different approaches can be taken. Others with extensive practical experience within this area have engineered a broad range of computer vision techniques that from the look of it applies to problem posed in the final problem statement. Since it is highly unlikely that any new computer vision techniques will be pioneered through this project the following test's main purpose will be to gain some hands-on experience, but also to figure out which system should be used for the final product. In a nutshell this test will not so much be testing the limitations of the tested techniques but shed light on what is realistic to implement at a relatively inexperienced skill level and within the timeframe set for the project.

3.4 TEST OF SIMPLE DYNAMIC AND STATIC SYSTEMS

It has become apparent that several relevant computer vision techniques exist for this project. What is needed now is some hands-on experience with the most relevant and straightforward techniques.

As mentioned in "3.3.6 Relevant Techniques to be Used", two systems will be tested to determine which direction the rest of the project will take. The two systems differ by being dynamic and static and by the end of this chapter it should be possible to choose between the two. Relevant criteria from the "2.6 Success Criteria" will be used to determine the success of each system. However, these will be simplified, since these tests do not necessarily have to fulfill the same kind of criteria as the final system.

This test will be divided into the Basic Test, which consists of two sections (the Image Test and the Video Test), and the Stress Test, an optional, conclusive test.

1) **The Basic Test**
    (1) **The Image Test**
    Instead of starting with a video feed, this test will be performed on two or more still images, to keep everything as simple as possible. The still images are supposed to simulate a very short excerpt from a video. For the static solution the difference between the two images should help determine the position of a cyclist and a car. For the dynamic test the environment will also be moving, since this system would be mounted on the car.
    (2) **The Video Test**
    This test is a continuation of the Image Test, and it tests both systems on a basic level with a video, just to see if it is possible to convert the systems made for the Image Test to work with video. If one of the systems fail here, the other system can already be chosen and no further testing will be necessary. If both systems succeed a stress test may be performed to determine the final solution for the project.

The criteria for the Basic Test are:

For The Simple Dynamic System:

- The cyclist is detected and tracked in the images, i.e. the algorithms can tell us where the cyclist is from frame to frame.

For The Simple Static System:

- Both involved objects (cyclist and car) are being detected and tracked from frame to frame.

2) **The Stress Test**
    An optional test, which may be performed to determine what the final solution for the project should be, depending on the results from the Basic Test.

If a stress test is conducted, stricter criteria will be necessary. Since these criteria would be based on the results from the Basic Test, these cannot be listed now.

As mentioned in "3.3.6 Relevant Techniques to be Used" the techniques and algorithms used for each system, will be the ones that makes it possible to fulfill the criteria in the most straightforward way and with techniques which possibly have some basis in what we have already learned. The Simple Dynamic System will use feature-based optical flow in the form of the Lucas-Kanade algorithm, explained in "3.3.5.3 Optical Flow", and for the Simple Static System the overall collection of techniques used will be those covered by BLOB Analysis, explained in "3.3.4 BLOB Analysis".

3.4.1 THE SIMPLE STATIC SYSTEM

The Image Test

The Simple Static System will here be tested with a few images, simulating a short excerpt from a video. Still images from a multimedia imaging portal were found for the test [USC]. The first, static image can be seen in Figure 17. It depicts two vehicles moving toward each other and is used as a substitute for the real-world scenario, which the system must work within, i.e. a cyclist and a car in a right turn situation. The reason for using this image is that it was not possible to find suitable images of a real-life right turn situation.

Figure 17: Image for the Image Test for the Simple Static System [USC].

The source code for the techniques for this test is largely based on the techniques described in "3.3 Computer Vision Techniques" and will therefore be only be described briefly here. The initial goal of this test is to turn the two vehicles into BLOBs, and if possible subsequently track them. The purpose is to see if a BLOB found in one image can correctly be found in the next image, when it has moved. That means there are three parts to this test:

- BLOB segmentation, where a binary image with BLOBs for the Independently Moving Objects (IMO) is created from the test image in Figure 17.
- BLOB Extraction, where the individual BLOBs are found and indexed.
- BLOB Classification, where the BLOBs are identified in their new position after moving.

These three steps are also the basis of creating a BLOB and then performing BLOB Analysis, as mentioned in "3.3.4 BLOB Analysis".

The first step in creating the static system was trying to isolate two useful BLOBs (the two vehicles as opposed to noise) by thresholding to get a binary image. Ultimately this was not an efficient way to isolate the BLOBs.

Instead background subtraction was done, and this gave a reasonable result. Background subtraction is done by subtracting the pixel values in the one image from the corresponding pixel value in the other image. When you have a static system with moving objects, a successful background subtraction should be able to remove everything but the moving objects, and this also worked for our system.

Figure 18 shows the result of using background subtraction and then a number of erosions and dilations, techniques mentioned in "3.3.3.2 Neighborhood Processing". Getting to this result, as seen in Figure 18, required multiple dilation and erosion steps which means that it did not seem to be able to run real-time if one wanted BLOBs without noise. This was discovered by using a timer function to profile the erosion and dilation steps, and they each took ~50ms to be calculated and around five to seven iterations were needed to achieve what is seen in Figure 18. Still, this was not very relevant at that point, since this was only a minor test and in a final system there would be time to optimize these kinds of things.

Figure 18: After background subtraction, erosions and dilations.

The next task was being able to track the individual BLOBs over several frames, i.e. if the BLOB on the left has moved downwards in the next image, it would be important for us to be able to know that this is the same BLOB. Since the final Static System would cover a known area, e.g. an intersection where the location of the road and bicycle path in the image is known, one easy solution would be to just assume that whatever BLOB is in the area of the image with the bicycle path, is a cyclist. Likewise, one could assume that a BLOB moving in the area of the image, where we know a road is, could be identified as a car or some other vehicle. If these assumptions were made, the system could assume a collision is at hand, if BLOBs move in a certain direction.

However this is not a very viable solution, and instead we tried to classify the BLOBs, so we could identify old BLOBs as the same BLOB in new images. This might also help us calculate the speed of the BLOB (and thus, the cyclist or the car). This is where the BLOB Extraction and Classification came in. To extract the BLOBs the Grass-fire algorithm is used. This worked fine and we now had an index number for each BLOB. The next step is classifying the BLOBs.

Classifying the BLOBs is done by extracting and matching features. One feature is the assumption that a BLOB in one image is the BLOB that have the position closes to that BLOB's position in the next image. The position of a BLOB is usually found by calculating its center of mass, done like this:

```
for all pixels in BLOB
     total_x += pixel_x;
     total_y += pixel_y;
            divide total_x and total_y with number of BLOBs
```

Code Block 2: Pseudo code for center of mass.

A qualified guess at which of the new positions that is the same BLOB can now be done using a similar approach as described in pseudo code below:

```
for all blobs
     for all new blobs
             compare distance to old blob
             shortest distance is the old BLOB's position
```

Code Block 3: Pseudo code for guessing new BLOB position.

The result of the above can be seen in Figure 19. This way of checking is not optimal for big amounts of BLOBs due to the algorithmic complexity of O(n^2)[7]. This however is not relevant as there are usually are so few BLOBs that it is not at all demanding compared to e.g. looping through all pixels in an image.

Furthermore, this is a fairly naive way of calculating new positions, as it would e.g. cause issues in a case with a different amount of BLOBs in two frames. Another issue is if two BLOBs switch positions due to them having directly opposite velocities, which would confuse this simple system. This could therefore be improved by also taking into account the velocity of the BLOBs and then using this to enhance the guess.

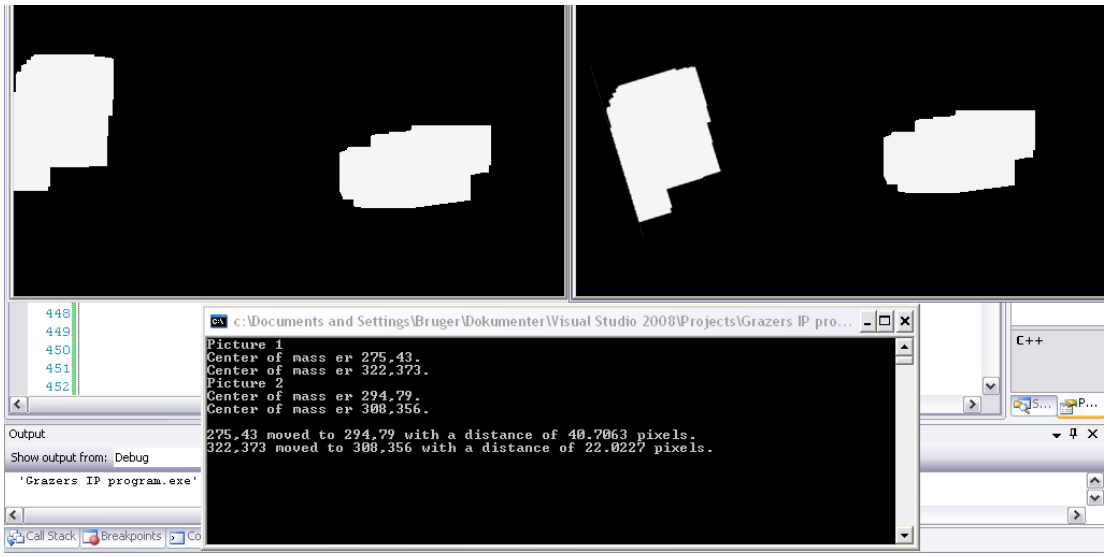


Figure 19: Using center of mass to guess new BLOB position.

Another approach would be to take the size or the shape or the BLOB into consideration, i.e. looking at other features in the BLOB Classification part. However, with the current system it was possible to find the same BLOB over several images, so it is viable to think that it would be able to do the same with a cyclist and a car, e.g. at an intersection where the location of the bike path and road, in the video's frame, is known. More importantly, this minor test of a static system fulfils the criteria of detecting the two vehicles over several images

The Video Test was also completed. However this gave almost identical results to the Image Test. The video used was simply put together with the same kind of still images used in the Image Test. The only difference in the source code was the way the image data was imported, instead of importing an image, a *while-loop* simply looped the video, showing a new frame with each iteration of the loop. Because of the minor differences between the Image Test and the Video Test, an entire chapter for it was deemed unnecessary. To conclude, this test showed that BLOB Analysis can be used to isolate and track objects, in this case two vehicles substituting for a cyclist and a car. Some limitations were found, but this test's success criteria were fulfilled and a lot was learned.

[7] This means that the number of calculations increase exponentially with the amount of blobs

The next chapter, "3.4.2 The Simple Dynamic System", will explain the Analysis Test for the dynamic solution and then one of the systems will be chosen.

3.4.2 THE SIMPLE DYNAMIC SYSTEM

When starting work on The Simple Dynamic System for the Analysis Test an attempt was first made to find existing example code. When looking for example code, a very thorough one was found, made by David Stavens [Stavens07]. This example estimated optical flow with the Shi-Tomasi algorithm and the feature-based Lucas-Kanade method, i.e. all that seemed to be needed and it seemed like a very simple and easily accessible approach, and therefore this example will be used for the the Basic Test. The example was also able to run with video from the beginning, so there was no reason to test this system with one or a few images, like it was done for The Simple Static System in the Image Test. The following sections will go through how the code works, this will be done more in-depth than for The Simple Static System's test since the techniques required here are significantly more advanced and are not described thoroughly in previous chapters.

Given a set of points in one image, optical flow is able to find those points in another image. This is the basic idea of optical flow. More specifically, given point [ux, uy]T in image I_1 find the point [ux + ϑ x, uy + ϑ y]T in image I_2 that minimizes ϵ :

$$\varepsilon(\delta_x,\delta_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} (I_1(x,y) - I_2(x+\delta_x, y+\delta_y))$$

Figure 20: Estimating optical flow.

Good Features to Track (Shi-Tomasi)

The feature-based Lucas-Kanade method works by finding old features in new video frames. When finding features that are good to track, they need to either have texture (ambiguity in tracking) or corners [Stavens07]. One effective way to detect these features is by using the Shi and Tomasi (or Kanade-Tomasi) corner detection algorithm, which is already built-in in OpenCV.

If no corner can be located, we have what is called the "aperture problem", which can be demonstrated by thinking of a barber pole. On this rotating pole, the lines are moving to the left or right, but optical flow thinks the lines move upwards or downwards [Murakami04]. The feature-based Lucas-Kanade function in OpenCV has a built-in feature to help with this problem, explained in the *"Pyramid"* section below.

Detection of Features in Different Frames (Lucas-Kanade)

For the calculation of the optical flow itself Stavens uses what is called the Pyramidal Lucas-Kanade method. The Lucas-Kanade algorithm concerns the detection of the same feature in two different frames, and the pyramidal part concerns an improvement of the algorithm.

To make the Lucas-Kanade method work, some assumptions are made. The first is called brightness constancy, and it assumes that brightness in a small region is the same, even if the location changes, i.e. the image moves. It is also assumed that the image motion of a surface patch changes gradually over time, this is called temporal persistence. The last assumption is spatial conference, which assumes that "neighboring points in the scene typically belong to the same surface and hence typically have similar motions." and "Since they also project to nearby points in the image, we expect spatial conference in image flow." [Stavens07].

Pyramidal

The "Pyramidal" part covers a technique used to improve the algorithm, when the motion is too big. This is done by reducing the resolution of the image a number of times (one time for every pyramid that you choose to use), and then measuring the motion in these sections. This feature is also built into OpenCV. If the aperture problem is encountered, it is possible to solve it by adjusting a parameter in the Lucas-Kanade function, more specifically the window size of the pyramid.

Shi-Tomasi in OpenCV

Stavens' code only uses two OpenCV functions to create the optical flow, with some support code. In the code, Stavens starts out by capturing a video file, he retrieves some information about the video, and starts a loop of the frames, i.e. he plays the video. Then he applies the Shi and Tomasi algorithm with the OpenCV function *cvGoodFeaturesToTrack()*, which takes several inputs:

```
void cvGoodFeaturesToTrack(
    const CvArr* image,
    CvArr* eig_image,
    CvArr* temp_image,
    CvPoint2D32f* corners,
    int* corner_count,
    double quality_level,
    double min_distance,
    const CvArr* mask=NULL,
    int block_size=3,
    int use_harris=0,
    double k=0.04 );
```

Code Block 4: Prototype for the Shi-Tomasi algorithm in OpenCV.

The required inputs are: *image* is the input image, *eig_image* and *temp_image* are temporary workspaces for the algorithm, *corners* is an output parameter which will contain the feature points, *corner_count* determines the amount of features to be found, the two doubles determine the quality of the features and the minimum distance between features, ( ) and *mask* determines if the entire image should be used (if NULL the entire image is used). The remaining parameters are optional.

Lucas-Kanade in OpenCV

The next OpenCV function is *cvCalcOpticalFlowPyrLK()*, and this calculates the Pyramidal Lucas-Kanade optical flow. The prototype for the function looks like this:

```
void cvCalcOpticalFlowPyrLK(
    const CvArr* prev,
    const CvArr* curr,
    CvArr* prev_pyr,
    CvArr* curr_pyr,
    const CvPoint2D32f* prev_features,
    CvPoint2D32f* curr_features,
    int count,
    CvSize win_size,
    int level,
    char* status,
    float* track_error,
    CvTermCriteria criteria,
    int flags );
```

Code Block 5: Prototype for the Pyramidal Lucas-Kanade algorithm in OpenCV.

The inputs: *prev* contains the first frame with the known features, *curr* is where we want to find the first frame's features, *prev_pyr* and *curr_pyr* are workspaces, *prev_features* are the features from the first frame, *curr_features* are the outputted locations of those features in the second frame, *count* is the number of features in the curr_features array, *win_size* is the size of the window to use to avoid the aperture problem, *5* is the maximum number of pyramids to use (0 would be just one level), the element of *status* is non-zero, if a feature was found in previous frame, *track_error* is used similarly, *criteria* determines for how long the algorithm should be run (determines speed and accuracy).

In Staven's example code, arrows are drawn where features are found, and the entire video is outputted to a new window. Since the length and direction of the arrows denote the optical flow and the criteria for the test is to found the relevant object in each image (e.g. a cyclist), the arrows of a certain length or in a certain direction will probably be able to show where it is.

The Basic Video Test

The test was done on a 3D-rendering of a moving road, seen from the front-view window of a car. The images were provided by [Vaudrey08] and were turned into a video. The reason for using this data is that a synthetic environment increases the possibilities of determining the quality of the output. The car on the right is used as a substitute for the cyclist, which the system should be able to detect in the real world.

Running the video with minor color adjustments from the example code yielded the following results:

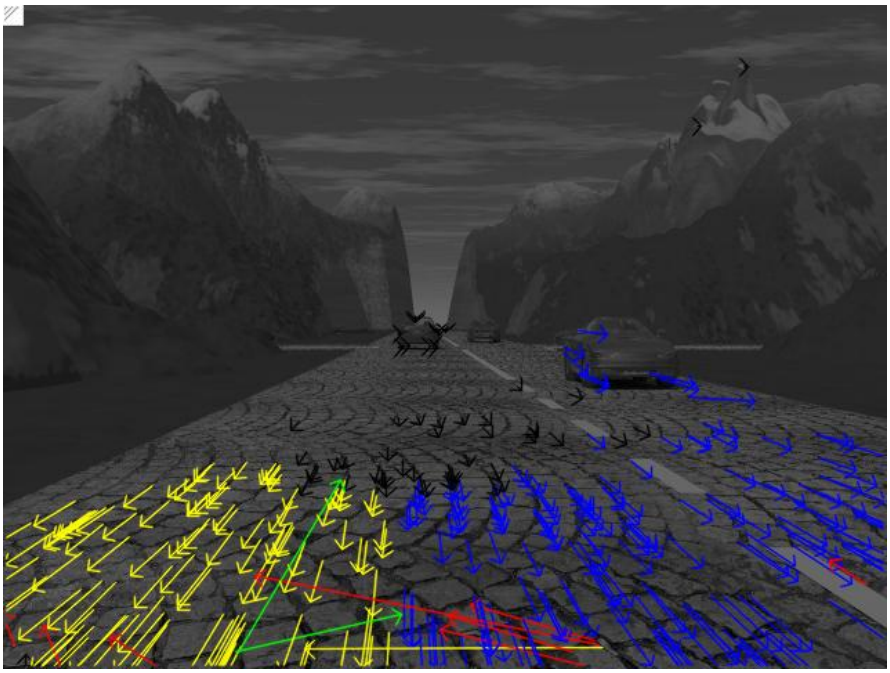


Figure 21: Sparse Lucas-Kanade in 3D environment (low amount of iterations).

It should be noted, that the arrows do not denote the precise start and end points of the movements. The start point is true, but the length of the arrow has been tripled, making it easier to see the direction. The colors denote different directions, making it easier to distinguish between them. There are obviously a lot of errors, i.e. the algorithm finds incorrect features and makes arrows all over the place in the bottom of the screen (due to the details on the road). Changing the parameters of the function to reduce these arrows increases computational time, but in this 3D rendered environment increasing the algorithm to its maximum amount of computations did not create a significant decrease in run time for the video.

Figure 22 is the same frame with the algorithm doing more iterations to increase the quality of the features, which are found. Notice the decrease in errors compared to Figure 21.

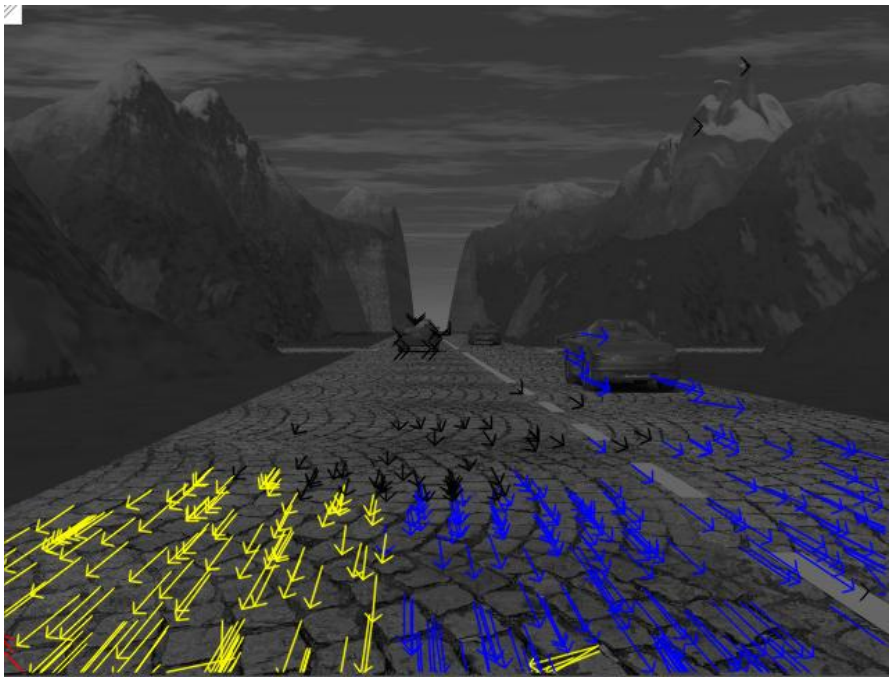


Figure 22: Sparse Lucas-Kanade in 3D environment (high amount of iterations).

At this point it is already possible to declare the Basic Test for The Simple Dynamic System a success, since the moving object (in this case, the car) has arrows on it, which means that the algorithm finds its features.

The next step would be to distinguish between the optical flow of the object and the optical flow of the background, e.g the road. The way this can be done is by comparing the vanishing point from the background with the vanishing point for the car. Since these differ, it should be possible to isolate the car. This however is not necessary to declare the Basic Test for The Simple Dynamic System a success, since the success criteria are fulfilled. A lot of knowledge was also gained from doing the test, and that was also one of the main goals of the test. Next, the static or the dynamic solution will be chosen and be the focus of the rest of the project.

3.4.3 TEST CONCLUSION

The results from the Basic Test indicate that both the static and dynamic solutions work within the simple success criteria defined in the beginning of "3.4 Test of Simple Dynamic and Static Systems", because the moving objects were detected.

At this point it could be relevant to perform a stress test as suggested in "3.4 Test of Simple Dynamic and Static Systems". However, it is not certain that the Stress Test would reveal specific flaws in a dynamic vs. static solution that would make the choice between the two obvious. It also seems limited how much more we could learn from such a test. Furthermore, it may prove difficult to determine whether any weaknesses that the Stress Test would reveal would be accredited to weaknesses in the implementation, the applied techniques or the dynamic and static solutions themselves.

Overall, the time seems better invested in moving forward and making a choice between the Static System or the Dynamic System based on our own preference. In short, the static solution seems simpler than the dynamic solution. The dynamic solution seems to offer more variety and even though it seems to be a bigger challenge, it still seems possible to implement it successfully. Since an important purpose of this project is to learn more

about computer vision and image processing, it makes most sense to go with the more challenging of the two: A dynamic solution is chosen as the focus of the rest of the project.

The following chapter lists the requirements for the system, which will be the basis for *4* Design.

3.5 REQUIREMENT SPECIFICATION

3.5.1 INTRODUCTION

3.5.1.1 PURPOSE

This requirement specification is based on IEEE standard 830-1998 [IEEE98] and [Biering-Sørensen88] in regards to the software product while process requirements are based on SPU-UML (Structured Program Development - Unified Modeling Language) [Hansen05]. This specification will list the functionality, interfaces, performance and attributes required to develop the product.

3.5.1.2 REFERENCES

The specification is backward traceable meaning that all requirements will be referred back to previous studies in the report. Thus when a requirement is stated a reference to the place of origin is stated as well.

The specification is forward traceable which in practice means that all requirements has a unique identification so it is possible to make direct references.

3.5.1.3 READING GUIDE

The requirement specification is written in natural language opposed to structured or formal language. It will be divided into the following four sections.

1. Introduction
2. System description
3. Specific requirements
4. External interface requirements

A test of how well the requirements are matched will be tested in a final test, see *"6 Test"*.

3.5.2 GENERAL DESCRIPTION

3.5.2.1 SYSTEM DESCRIPTION

The purpose of the system is to prevent right turn accidents between cars and cyclists in the Danish traffic.

In short terms this will be done by detecting cyclists in situations described in the traffic analysis part "3.1 Analysis of Right Turn Accidents". If an unsafe situation is detected the system has to be able to give an output as mentioned in "3.5.2.1 System Description" as a console print out.

How to warn the involved parties is not a part of the product, as first mentioned in "2.5 Delimitation", and will therefore not be a part of the system requirements. To provide an overview over the system a context-actor diagram is produced, see Figure 23. This gives the highest level view over the input and output from the involved actors. The system receives input about the position of the two actors in risk of colliding. The combination of the information from the two input actors makes the system capable of detecting a cyclist.

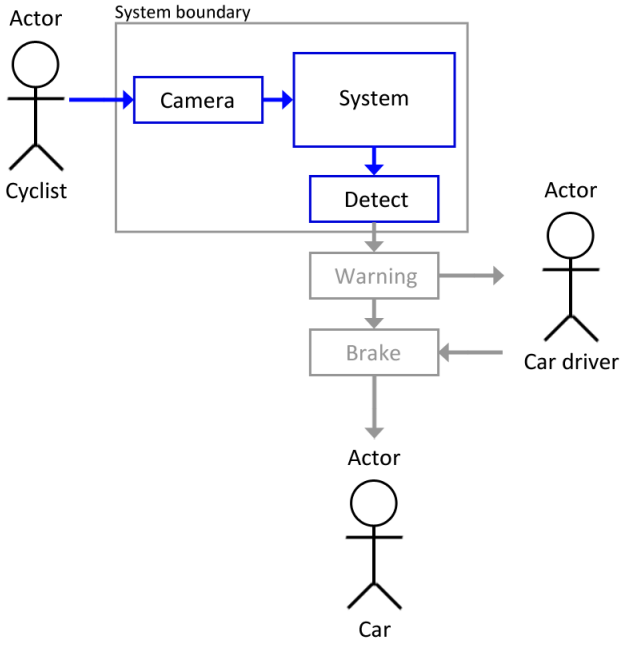


Figure 23: Overview of the system with a context-actor diagram.

The hardware part of the system consists of a camera located on the car with the field of view covering the right side of the car. The camera is static and provides footage which is transmitted to a computing unit running the product. The software part in this system consists of a program that is able to detect a cyclist moving alongside the car.

3.5.2.2 THE FUNCTIONS OF THE PROGRAM

The program's main function is to prevent collisions between cars and cyclists in right turn situations.

To be able to detect the cyclist the program has to be able to obtain information about which direction objects within the field of view moves. The program has to be able to tell whether the cyclist moves at a speed relative to the vehicle which makes the cyclist a possible threat leading to a warning print out.

3.5.2.3 LIMITATIONS OF THE PROGRAM

The system will only be able to detect the cyclist. It will not be involved in warning the driver or automatically applying the brakes beyond executing a print out as mentioned in "3.5.2.1 System Description".

3.5.2.4 THE FUTURE OF THE PROGRAM

The product developed for this project is a prototype to be used for research purposes only. A final commercial product with the same purpose will technically be distinctly different. This means that the initial prototype should not be used as a direct base for a future commercial program. Therefore the development of this first program should not take possible future extensions into consideration.

3.5.2.5 USER PROFILE

The user of the system is the module that takes care of the warning part.

3.5.2.6 DEVELOPMENT PROCESS REQUIREMENTS

The development process will be supported by the use of SPU-UML. An iterative development model is chosen which both deal with the design and implementation of the program.

3.5.2.7 SCOPE OF DELIVERY

The development model chosen is a modified version of the w-model stated in [Hansen05].

The model exists of five modules (one design module and four implementation modules), one internal deadline, three internal tests, and a final test just before the external delivery date, see Figure 24:

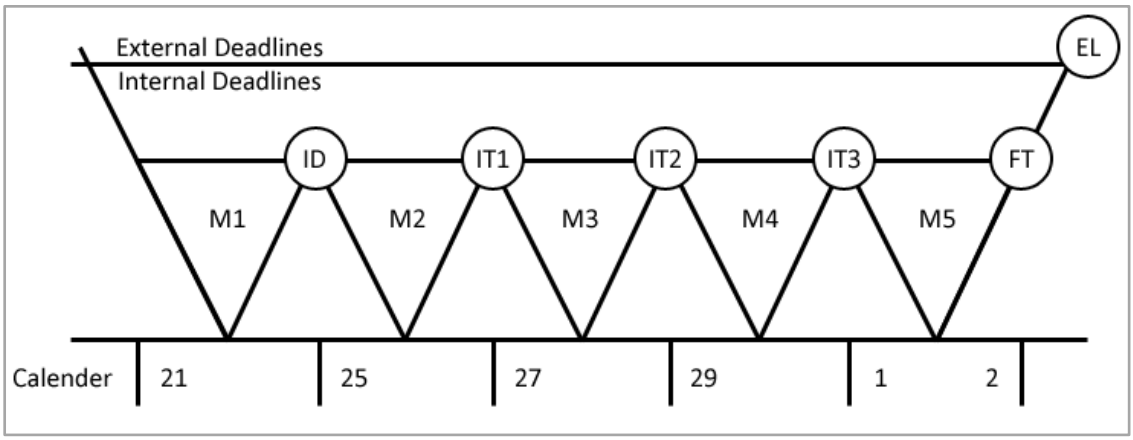


Figure 24: Modified W-model for delivery of the program.

Module review:
M1: Design of the system. List of functions and features which are going to be implemented.
M2: Basic program structures: Video input, output, first simple detection system.
M3: Improving detection feature.
M4: Finishing detection system.
M5: Tweaking and optimisation.

Delivery and tests:
ID: Internal delivery of finished design draft.
IT1: Internal test 1: Test of simple system.
IT2: Internal test 2: Test of detection-part.
IT3: Pre-final test: Training on recorded test footage.
FT: Final test.
EL: External delivery.

The development progress runs from the 21th of November to 2nd of December.

3.5.2.8 ASSUMPTIONS/PREREQUISITES

In order to have a working program some prerequisites need to be fulfilled. These prerequisites involves both hardware, software and knowledge about different right turn situations that results in collisions.

Camera, computer and what is needed to secure the camera properly to the vehicle are the hardware needed for the program to function. The camera is needed in order to collect image data from the surrounding traffic. The computer will be used to process these data in order for the bicycles within the camera's field of view to be detected.

The program will need daytime lighting conditions to function properly.

3.5.3 SPECIFIC REQUIREMENTS

3.5.3.1 DEFINITIONS

The following definitions are design rules for developing the program.

- The programming language is C++ as this is what OpenCV is optimized for.
- The delivery platform will be x86 compatible PC with 32bit Windows.

3.5.3.2 FUNCTIONAL REQUIREMENTS

To describe the functional requirements of the system use cases of general situations are defined. The functional requirements must be implemented for all use cases. First a brief overview of the use cases are presented here and then further down a detailed specification of each one is listed.

Use Case Overview

For these use cases two actors are involved, one being the cyclist and the other being the car driver. There are four use case scenarios of this system, first a use case diagram list to get an overview followed by a detailed description of each particular case.

Note that all use cases are simplified representations of real life situations as these are only used to break down the system in smaller parts to ensure an overview of the functional requirements. The use cases are based on "3.1.4 Where the Cyclist is Placed Compared to the Vehicle".

Use Case 1
Scenario: A cyclist is catching up to a car approaching a right turn situation in a cross section. In this case the front wheel of the bicycle is the first thing to enter system's field of view.

Use Case 2
Scenario: A cyclist is approaching a junction and the car is catching up to the cyclist. In this case the back wheel of the bicycle is the first thing to enter system's field of view.

Use Case 3
Scenario: A cyclist is within the camera’s field of view and travelling alongside the car. The car and cyclist is travelling at the same speed.

Use Case 4
Scenario: No cyclist is in the system's field of view.

An exception to these use cases also exist where the cyclist or driver ignores traffic rules and regulations and proceeds across the cross section. Because of the complexity of the scenario in this exception this use case is delimited from the program.

The system is divided into two functions:

- A function focusing on detecting the cyclist in more than 9 out of 10 situations.
- A function focusing on preventing the system from wrongfully detecting a cyclist, when there is no cyclist, in less than 1 out of 10 situations.

The first function is implemented for Use Case 1, 2, and 3, while the second function is implemented for the Use Case 4.

The system is expected to be able to accommodate for situations where several cyclists are in the field of view as it should take only the most dangerous one in consideration. These will not be considered specific use cases simply because of the near infinite combinations of use cases it would bring, but will instead merely fall within the already defined use cases.

Any non-cyclists will be considered noise and to fall within the already defined use cases. In other words: If a cyclist is catching up t the car (Use Case 1), and pedestrians are in the field of view, the case will still be categorised as Use Case 1.

Function 1: Detecting the Cyclist

The purpose of this function is to detect a cyclist in the system's field of view.

Requirement: Detecting the cyclist in more than 9 out of 10 situations.

Input: A video recorded by a camera placed on the car. The video is recorded in daylight conditions as this is when the accidents most frequently happen as found in "3.1.3 When the Collisions Happen".

Output: When the cyclist is detected the system executes a console print-out.

Properties of involved parties: If the involved parties have the properties mentioned below, then the system has to detect the cyclist.

Cyclist - a moving object with certain properties:

- Velocity: The cyclist is moving with a speed between 0 km/h to 45 km/h, see section "3.1.5 Speed and Reaction Time".
- Placement: The cyclist is placed at the right side of the car on a bike path or sharing the road with the car, see section "3.1.4 Where the Cyclist is Placed Compared to the Vehicle".

Car - an object that moves along with the camera and has certain properties:

- Velocity: The car turns with a speed of maximum 40 km/h, see section "3.1.5 Speed and Reaction Time".

Function 2: Detecting Only the cyclist

The purpose of this function is to avoid detecting a cyclist, when no cyclist is in the camera's field of view.

Requirement: Having less than 1 out of 10 false positives when no cyclist is in the camera’s field of view.

Input: A video recorded by a camera placed on the car. The video is recorded in daylight conditions as this is when the accidents frequently happen as found in section "3.1.3 When the Collisions Happen".

Output: Nothing.

Properties of involved parties: If the involved parties have the below mentioned properties then the system should not detect a cyclist.

Car - an object that moves along with the camera and has certain properties:

- Velocity: The car turns with a speed of maximum 40 km/h, see section "3.1.5 Speed and Reaction Time".

3.5.4 External Interface Specifications

3.5.4.1 User Interface

This system does not have a user interface as it functions without any direct input from the driver or cyclist.

3.5.4.2 Performance Requirements

The system needs to perform well enough to be able to detect the cyclist soon enough to avoid a collision.

4. DESIGN

With the Requirement Specification done, it is now clear what features the system needs be able to support. Now it is necessary to figure out how to design a system that fulfills those requirements. This is what this design chapter will be about, basically taking the system from 'what it has to do' to 'how it has to do it'. The overall theme of the required system is 'motion detection' and more specifically 'detection of Independently Moving Objects' (IMO detection) as this is what the key features from "3.5 Requirement Specification" relates to.

In the end of the analysis a test found that a dynamic solution is preferred for this project. The test showed that a dynamic system could be a viable solution when combined with optical flow.

In order to ensure that all possibilities are thought of an attempt will be done to uncover whether there are any alternative techniques that work better than the optical flow attempted in the analysis test. The area which will be looked into is however still only for dynamic systems.

When a technique has been chosen it will be looked into what algorithm this technique needs to be able to segment the wanted motion and achieve the detection of the wanted IMO.

When done with this chapter it should be clear exactly how to implement a system fulfilling the requirements.

4.1 DYNAMIC SOLUTIONS

In order to find a certain Independently Moving Object via a dynamic system, different solutions need to be examined so that the best possible solution can be chosen for implementation. The solution tested in "3.4.2 The Simple Dynamic System" was the estimation of optical flow, which is a method used to recreate the 3D information from a series of 2D images. Optical flow calculates the motion between a number of image frames, often just two, taken at two different times. This method is tested in the analysis section of the report with partial success, as it shows promise even though it was not developed fully. Within the area of optical flow there are multiple different ways of achieving the desired result, such as the sparse (i.e. feature-based) Lucas-Kanade method [Kanade81] used in "3.4.2 The Simple Dynamic System". Looking realistically at the time frame of this project, then it is a significant benefit to have tested optical flow because we now know that it is a technique that is technically feasible to implement in this project.

However, multiple alternatives to optical flow exist for the Dynamic System, one of which is template matching. As described in "3.3.3.2 Neighborhood Processing", this technique examines the image from the upper left corner to the bottom right corner looking for resemblance to a template image. The template image could in this case be a template of a cyclist. This technique is however hard to get entirely reliable as cyclists come in lots of shapes and colors.

Stereo vision, as described in "3.3.5.1 Stereo Vision", was another solution to find a cyclist. Stereo however should only be used if other techniques are not sufficient. The downside here is that another cameras is needed, which obviously adds a level of complexity. Also, this was used for a similar Medialogy project.

One way of recognizing IMOs is to segment the image into a 3D representation, similarly to the idea behind stereo vision and disparity maps. As described in this paper [Modrow07] a way to go about this is to project out an infrared grid and then extract spatial information based on the deformation of this grid. As the grid is infrared it will not be visible to humans and thereby not annoy anyone. This, however, requires that there is no other significant infrared interference.

While the methods mentioned above should contain relevant possibilities, then it is in no way certain all possibilities are exhausted. On the base of what has been covered in the above chapter it can be concluded that optical flow is a viable dynamic solution for this project and will be the technique which will be looked further into. This is mainly due to the fact that it is fairly straight forward to implement, it is known that it is a feasible solution, and there are many options within optical flow.

4.2 OPTICAL FLOW

Optical flow is an attempt to provide an approximation of the true motion field from a sequence of 2D images as described in "3.3.5.3 Optical Flow". Since there are many options within optical flow this section of the report will be looking into methods of optical flow which can fulfill the task of detecting the correct Independently Moving Objects. Two terms, dense and sparse, represent the overall approaches to optical flow.

Dense methods calculate optical flow for all pixels in an image, which means that in theory the dense methods estimate motion for every pixel in an image. In practice some areas are very uniform in their texturing and thereby it is not possible to find a plausible position for a given point in the next frame [Bradski08].

Sparse methods calculate optical flow on features only and thereby do fewer computations resulting in a faster estimation. Just as with dense methods sparse methods also rely on a certain amount of contrast and contours in the image. Features used in a sparse optical flow might include the IMO which has to be detected, however results from the analysis test show that it is not necessarily features from the IMO which are detected. Dense methods are generally more likely to include information about the IMO due to the additional pixels computed compared to sparse methods [Bradski08].

This means the dense methods show the most potential as they provide a better chance of tracking the IMO. With the choice of dense methods it is relevant to look into which dense optical flow implementation to choose for the final system.

4.3 CHOOSING A PARTICULAR DENSE ALGORITHM

When choosing a dense method it would be preferable first to take a look at which built-in functions OpenCV provides. In OpenCV 1.x there are three dense optical flow solutions built-in. The first one is Horn & Schunck [Horn81], which is a global variational method. The second is the block matching method which focuses on dividing the image into a suitable number of blocks and finding the corresponding blocks in the next image. The common denominator for the first two methods is the lack of accuracy.

For the Horn & Schunck method the issue is that it is a global method which implies global movement which again implies that small local displacements are not well tracked. This can be very harmful for the final system which aims to detect small moving objects (when the car and cyclist are far from each other) [Marzat09]. The downside of the block matching method is the lower resolution gained by the fact that the accuracy is a function of the size of the blocks. The resolution of the velocity information gained is the image size divided by the size of the blocks. To get an overview of the pros and cons of the different methods, see Table 4.

The local variational method of Lucas-Kanade is built into OpenCV. It is a dense optical flow method calculating the flow for every pixel (i.e. a different method than the Lucas-Kanade algorithm used in "3.4.2 The Simple Dynamic System"). This method has a better resolution than the block matching method and also has an advantage over the global method of Horn & Schunck, because of the benefits of being a local optical flow method: Lucas-Kanade makes use of an additional assumption that all motions are small and coherent at least in local areas of the image. This makes the algorithm interesting because of its robustness to noise and the fact

that the assumption about local motion makes the algorithm capable of tracking small motions in the image. In the analysis test the sparse Lucas-Kanade algorithm was implemented with pyramids. The reason to use these pyramids is as mentioned in "3.4.2 The Simple Dynamic System" that even within feed from a video camera running 30 fps large and incoherent motions are commonplace, so a large window is needed in order to track the motion, but with a large window the coherent motion assumption is broken.

Dense method	**Pros**	**Cons**	**Implementation order**
Lucas-Kanade	Included in OpenCV 1.x Capable of tracking small motions Robustness to noise	Large motion	1
Horn & Schunck	Included in OpenCV 1.x Iterations	Small local displacements are not well tracked Accuracy	2
Block Matching	Included in OpenCV 1.x	Small local displacements are not well tracked Accuracy Returned velocity image in low Resolution	3
Farnebäck	New method must imply improved properties. Implemented with pyramids in OpenCV	Only Included in OpenCV 2.0	4

Table 4: Pros and cons for dense optical flow algorithms in OpenCV.

Based solely on theory it is uncertain which of the above mentioned dense methods that will suit this project better. An implementation of all of them may prove necessary and therefore the implementation order is noted in Table 4.

The most promising optical flow method appears to be the dense Lucas-Kanade method. It has the advantage of being able to track smaller local displacement with a high velocity information resolution. This will be implemented without the pyramids as OpenCV does not support this part of the algorithm. This implementation process will be fast because of the built-in function *calcOpticalFlowLK()* in OpenCV.

4.4 CHOOSING WHAT TO BUILD ON TOP OF OPTICAL FLOW

With the optical flow algorithm in place it is now time to take a look at what needs to be done as the next step. The result of the optical flow algorithm is a set of displacement vectors for all pixels in an image, which describes the motion of that particular point compared to the previous frame. That is, we get an approximation of the true motion field. From this optical flow method it is necessary to find the relevant objects to track.

4.4.1 PHYSICAL SITUATION IN THE TRAFFIC

Before the information from the motion field is to be calculated, the first task is to determine which motion field needs to be observed.

The camera has to be mounted on the car as per choosing a dynamic system. But the question still remains what should be in the field of view of the camera and exactly where to place the camera. In order to answer these questions a detailed look at the generalized situation of a potential right turn accident from the traffic needs to be examined.

The first thing to be looked into is the exact motion of the car, and to do this one has to base the calculations on a certain kind of car. Instead of attempting to define what might be a viable car, it is decided to choose a specific car. This is an approach that makes sense since any finished commercial product would also have to be configured in some way to work on a specific car model, e.g. by supplying the specific data and then the system would adjust to these accordingly. In this section the car used is a 1997 Skoda Felicia since a such car is available for testing. The specific data used in the calculations follows from Table 5. Besides the data from the car some additional assumptions are made to simplify matters, see below. More on these assumptions in *7* Discussion.

Assumptions:

- The car and cyclist drives in the same directions.
- Distance between car and cyclist orthogonal on the driving direction is 0.5 m.
- Same velocity in the turn as just before turning.
- The car takes the right-turn along the line of a perfect circle.

Skoda Felicia 1997	[m]
Length	3.855
Width	1.635
Turning radius	5.250

Table 5: Measurements of the car, which is available for testing [CDb].

When viewed from a bird's eye perspective the scenario will look like the picture in Figure 25. The dangerous situation occurs when the cyclist and the car is located in the point *p* at the same time. The car is located in point *p* when the bonnet hits it until the car leaves the point with the rear end. The y-coordinate to the point *p* is found with the parametric form for circles and the fact that the x-coordinate to the point is known.

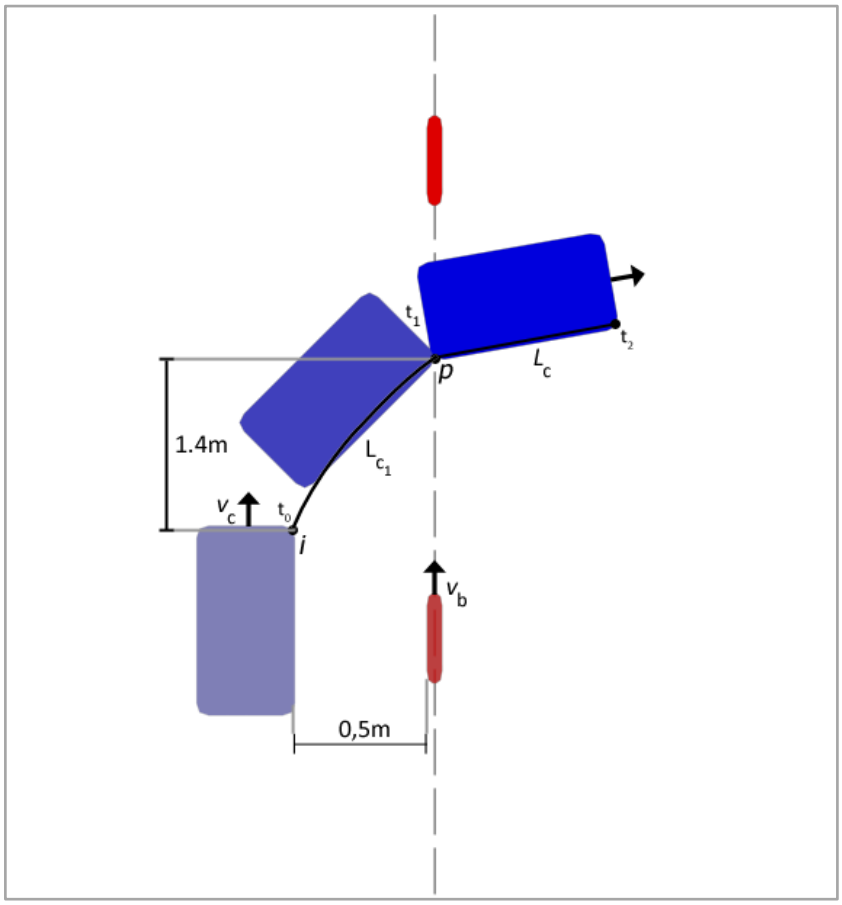


Figure 25: Bird's eye perspective on right turn situation.

The time $t_1[s]$ is the time it takes the car to go from *i* (time t_0) to *p* depends on the velocity of the car $v_c\left[\frac{m}{s}\right]$ and the distance $L_1[m]$to the point *p*. It looks like this:

$$t_1 = \frac{L_1}{v_c},$$

where L_1 is calculated by the fact that the arc-length from *i* to *p* is known.

The car will leave the point *p* again when the rear end has passed *p*. The time it takes the car to go from *i* to the rear leave *p* is denoted $t_2\ [m]$and calculated as:

$$t_2 = \frac{(L_1+L_c)}{v_c},$$

where $L_C[m]$ is the length of the car.

Now it is possible to calculate the interval where the system has to detect the cyclist in terms of where the cyclist is placed in relation to the car. If v_b is the velocity of the cyclist and $L_{c1}[m]$ and $L_{c2}[m]$ is the distance to the point *p* from where the cyclist is placed at $t_0$in relation to t_1 and t_2 then the critical region (detection interval) R_c is:

$$L_{c_1} < R_c < L_{c_2}$$

Where

$$L_{c_1} = (v_b - v_c)t_1$$

$$= (v_b - v_c)\frac{L_1}{V_c}$$

$$= (\frac{v_b}{v_c} - 1)L_1$$

and

$$L_{c_2} = (v_b - v_c)t_2$$

$$= (v_b - v_c)\frac{(L_1 + L_c)}{V_c}$$

$$= (\frac{v_b}{v_c} - 1)(L_1 + L_c)$$

This result shows that the distance between the car and cyclist where the detection has to take place in order to avoid a collision are dependent of the relative velocity between the car and cyclist, and only this. In Figure 26 and Figure 27 a 3D base area diagram is shown in order to see the relationship between the three quantities L_{c_1} or L_{c_2}, v_b and v_c. Note that the interval used for v_b and v_c are the ones found in "3.1.5 Speed and Reaction Time". v_b =] 0 km/h ; 45 km/h] and v_c = [5 km/h ; 40 km/h]. Minimum for v_b is set to 0 km/h where 0 km/h is not a part of the interval.

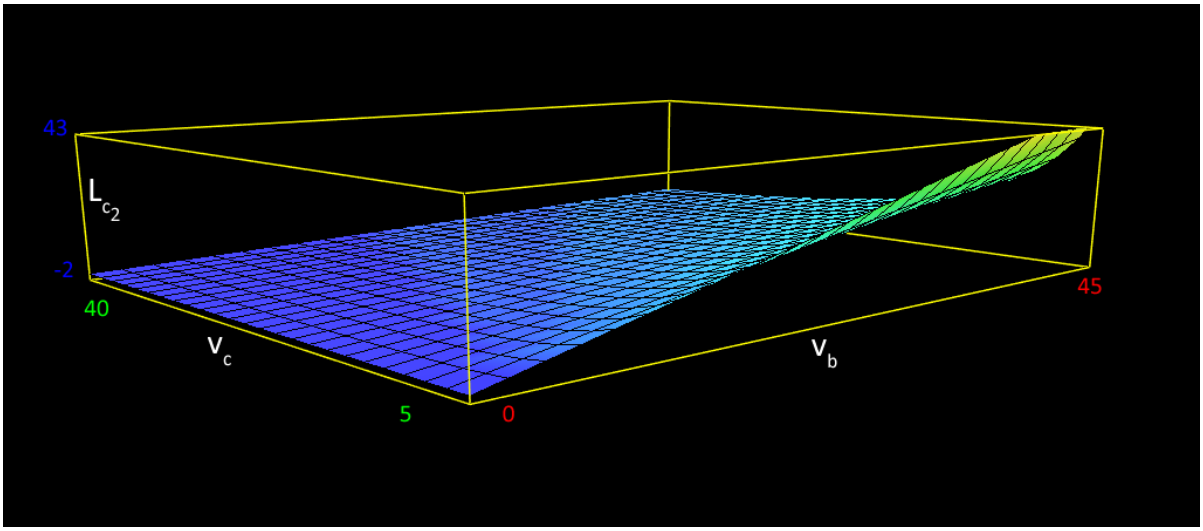


Figure 26: (a) 3D base area diagram of relationship between velocities of the car, cyclist and the critical distance from c_2 *to* p.

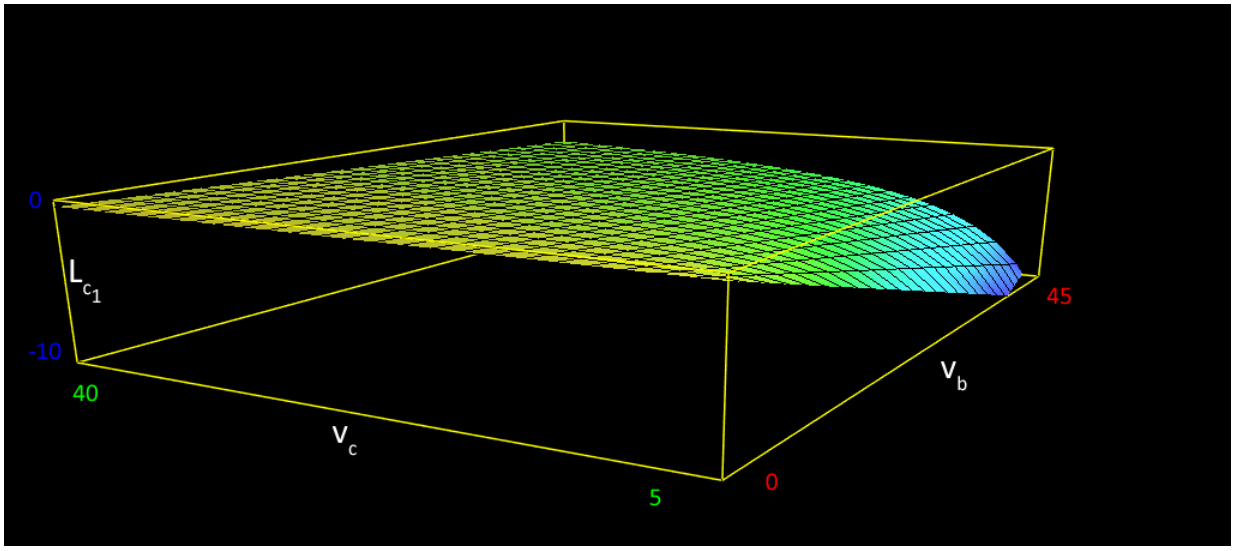


Figure 27: (b) 3D base area diagram of relationship between velocities of the car, cyclist and the critical distance from c_1 *to* p.

Figure 26 shows that the biggest value of L_{c_2} is obtained with the highest velocity of the cyclist and the lowest velocity of the car, which come as no surprise. This value is $L_{c_2} = 42.93\ m$ and is shown in Figure 28. Figure 27 shows that the lowest value is $L_{c_1} = 0\ m$ when the cyclist is not moving, regardless of the velocity of the car. In simple terms this means that if the cyclist is standing still then the cyclist needs to be at the point *p* in order not to be hit by the car. This is not entirely true due to the width of the car which is 1.635 m as shown in Table 5.

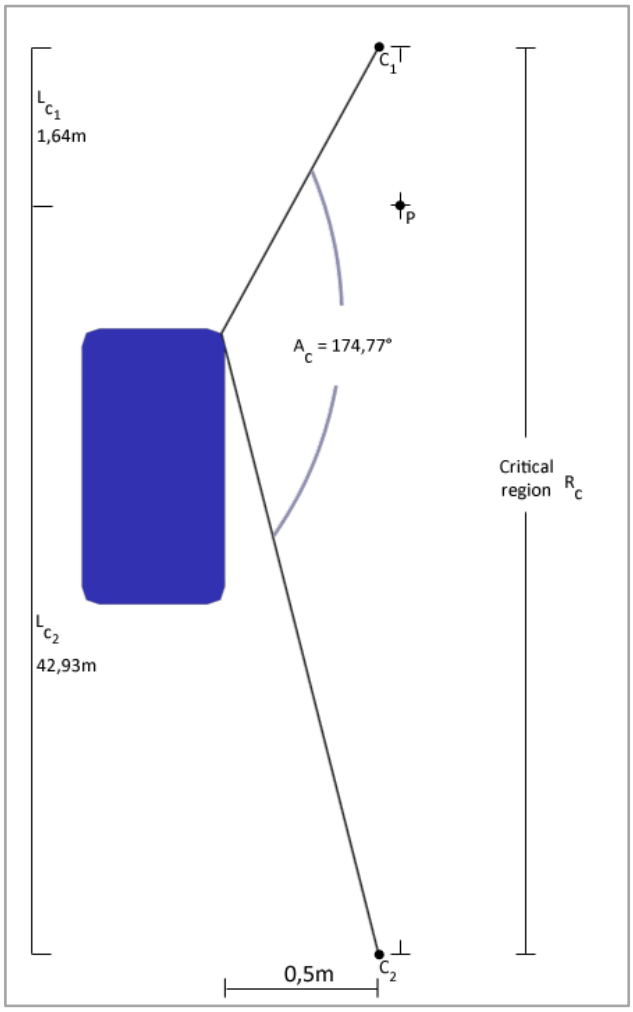


Figure 28: Angle needed to cover critical region.

Now it is possible to calculate the relative angle between points c_1 and c_2 Figure 28 in order to see the cyclist in the critical region. The situation is drawn in Figure 28 and with simple angle calculations it is shown that the horizontal angle of view has to be $A_c = 174.77^{\circ}$ in order to be able to overview the area in which the cyclist have to be detected. It might be necessary to have an even wider angle of view in order to have the cycle detected before it enters the critical region.

4.4.2 AMOUNT OF CAMERAS

The demand of a horizontal angle of view of 174.77˚ calls for the discussion on which camera to use and how many. The discussion about the choice of camera is also related to what kind of input the dense optical flow algorithm demands. This is on the other hand very hard to determine at this point, as it is decided in "4.3 Choosing a Particular Dense Algorithm" to pick the optical flow method which works better in the later implementation phase.

The conclusion on this discussion about camera choice is solely based on the horizontal angle of view result, and has to be reviewed when the requirements from the optical flow method are known.

The most simple solution to monitor the critical region is with one camera with an angle of view equal to 174,77˚. But even cameras with some of the smallest focal lengths at 13mm (35 mm equivalent) [Heikkila96] do not cover more than 108˚ horizontally. Three options could be a solution to the problem:

1) Make use of an ultra wide-angle lens, known as fisheye lenses.

2) An ordinary lens filming into an omnidirectional mirror.
3) Using several cameras with normal lenses.

The first and second solution generate non-rectilinear images (straight objects appear with straight lines in rectilinear images, in non-rectilinear they are curved). The non-linear images can be converted into rectilinear images so the motion field from these can be correctly extracted. This is how ever a more complex task compared to using multiple cameras to cover the critical region. An advantage from using multiple cameras is that each camera (and corresponding system) is able to look for different kind of motion.

As stated in the "3.5.3.2 Functional Requirements" four different use cases area part of the system and have to be fulfilled. The cyclist in Use Case 1 enters the critical region at point c_2 Figure 28 and the cyclist from Use Case 2 enters at point c_1. The cyclist from Use Case 3 where the cyclist and car have approximately the same speed is placed in the middle between the two points. Use Case 4 is not interesting in this context, as there is no specific motion to monitor.

The motion needed to be detected in Use Case 1 move from right-to-left in the image. If it is not moving in that direction, then it will never be in the risk of colliding with the car. The same is true for Use Case 2 only moving from left-to-right but with different sign. The motion needed to be detected in Use Case 3 is the one with little to no motion relative to the car.

In this way the system has to look for different kind of motion depending of what way it is looking. As seen in Figure 29 simple angle calculations show how it is possible to cover the critical region with the use of two cameras with traditional lenses with an angle of view on 50˚.

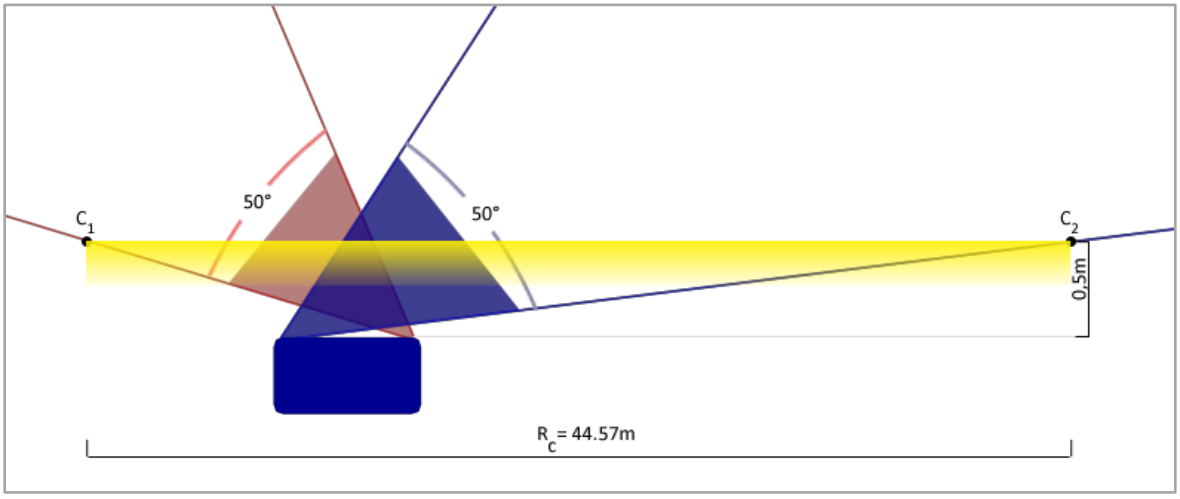


Figure 29: Coverage of critical angles.

The solution to cover the critical region with multiple cameras is chosen not only to simplify the system, but because of the possibility it offers to divide the overall system into smaller subsystems. One approach to designing and implementing a product within the limited time span given in this project (see "3.5.2.7 Scope of Delivery"), is to lay the framework for an overall system design, but only dedicate time to design and implement selected modules of this overall system.

For this reason it is chosen to go with the overall solution involving two cameras, but only focusing on the design and implementation of one of the two cameras. This also implies that not all use cases are going to be implemented in the system. Following next is the deliberation over which of the two cameras to choose.

4.4.3 CHOOSING AN USE CASE

Since the results from "3.1 Analysis of Right Turn Accidents" respectively cover trucks in Denmark and cars in the US, the data is not seen as being representative for right turn accidents with cars and cyclists in Denmark. Therefore these data are not used to make an argument for choosing one of the use cases. It also seemed apparent that accidents do occur for both Use Case 1, 2 and 3, so the system will be helpful regardless which use case is chosen.

Due to the different distances from the car to the critical points c_1 and c_2 (see Figure 28) the task to detect the cyclist in Use Case 1 and 2 have different degrees of difficulty. A farther distance to the car gives less movement in the motion field, and this is why Use Case 1 in respect to trackable movement is probably more difficult to implement successfully.

Still, at this point many things have been delimited, and if a system which works for Use Case 1 is successfully implemented then it can be argued, that it is also possible to successfully create the systems for the other use cases. Thus, a solution for Use Case 1 (i.e. a cyclist catching up to the car) is chosen, and the system will be created with a backwards facing camera, as seen in Figure 30.

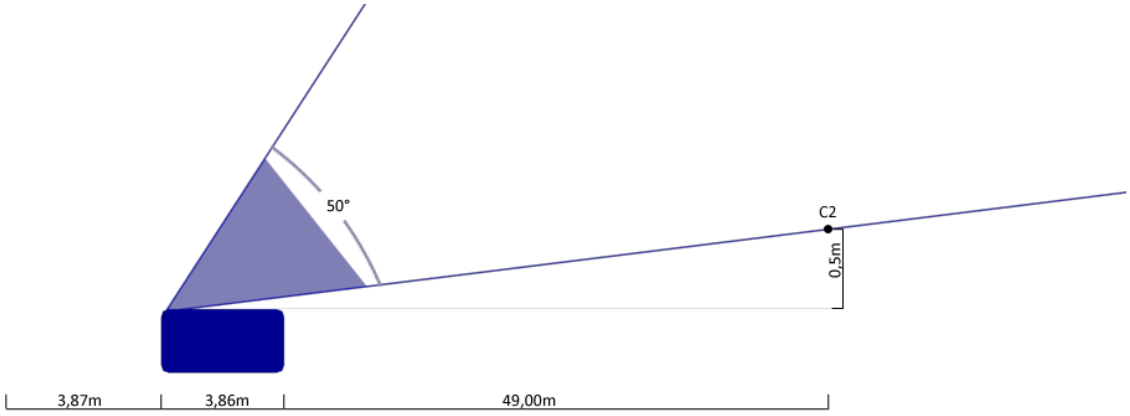


Figure 30: Coverage with rear camera.

4.4.4 DESIGN OF THE DETECTION PART

The task of ending with a system which gives a console print-out when a cyclist is in the critical region involves a number of steps on top of the fundamental optical flow algorithm. The subdivisions are as follow:

1) Prerequisites.
2) Preprocessing.
3) Identification of IMO.
4) Deciding if IMO is in danger.

4.4.4.1 PREREQUISITES

As stated in "3.5.3.2 Functional Requirements" the input to the function that need to detect the cyclist is a video feed. The video camera acquiring the video feed has certain properties, which should correspond with what is needed in order to make the detecting as qualified as possible.

Frame rate: The optical flow methods we choose to start implementing are not good at tracking large motions which would be solved by using the principle of pyramids. This requires the frame rate to be as high as possible seeing as the motion will be smaller with the same physical 3D velocity in the real world. In this project it is unrealistic to get hold of a high speed camera and at the same time it is unrealistic that at system based on a

high speed camera will be able to process the information that fast seeing as how a dense optical flow calculation between to frames is a time consuming task. A camera with 25 fps is the compromise between desirable high frame rate to satisfy the optical flow algorithms and a low frame rate to keep the performance requirements acceptable.

Image sensor: The better quality of the sensor and likewise image quality, the better performance of the optical flow algorithms as they rely on finding edges, corners and other significant patterns in the images. The use of a dedicated sensor for red, green and blue light (3CCD) might be favorable, as the quality of the images improves. The use of webcams with a low quality sensor is precluded.

Image stabilizer: As the camera will be placed on the car and thereby be impacted by of the natural vibrations from driving an image stabilizer may be necessary. The stabilization could be done later in the process by the software but this will impact performance to some extent, making in-camera image stabilization more desirable.

Focal length: The field of view is determined by the focal length and sensor size in a video camera. From Figure 29 it is shown that the critical region on the right side of the car can be covered using two cameras with a angle of view of 50 %. This corresponds to a focal length on 40 mm measured as 35mm equivalent.

With the properties of the camera in place, the position of the camera needs to be determined. In order to obtain optimal functionality in the system it is important to know where the camera should be placed on the vehicle so that the cyclist is detected from point c_2 until the forward facing camera takes over. If the two cameras are placed in each end of the car pointing towards each other as showed in Figure 29 this gives the best coverage in the region where the cyclist will appear just beside the car.

The camera in this system should therefore be placed in the uppermost front right corner of the car as seen in the illustration below, Figure 31.

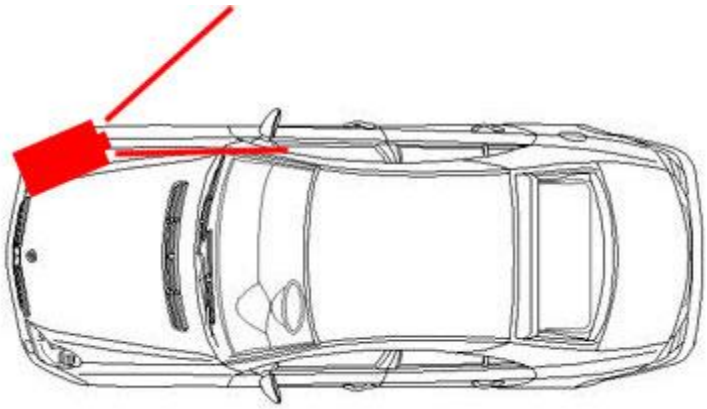


Figure 31: Best placement of camera for Use Case 1.

4.4.4.2 PREPROCESSING

Due to the cameras fixed image ratio some sections in the picture might not provide the system with any useful input about the IMO. In the horizontal direction all of the image will be used, as the 50˚ angle of view is needed in order to cover the critical region. But especially in the right side of the images where the cyclist is far away and occupying a small amount of pixels, there are areas where the cyclist will never be placed, see Figure 32. This calls for the use of ROI (explained in "3.3.2 Pre-Processing") or cropping undesired parts of the image away before the optical flow algorithm takes over. The run-time speed is significantly improved by reducing the

amount of pixels the optical flow needs to go through by two. So it makes sense to define a ROI as marked by the red line in Figure 32

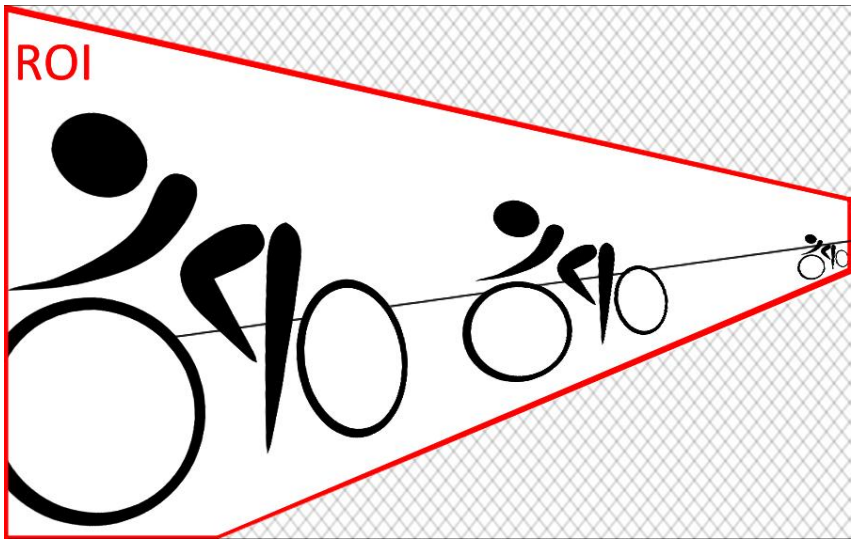


Figure 32: ROI for cyclist.

Some of the OpenCV built-in optical flow algorithms take gray-scale images as input. If this is the case the images captured from the video stream will be converted to grey-scale.

All of the optical flow algorithms need clearly defined structures to be able to track the motion. It may prove interesting to experiment with the use of sharpen techniques to help improve the robustness of the optical flow methods. One example to such an approach is the Gaussian Sharpen kernel, see more in "3.3.3.2 Neighborhood Processing".

4.4.4.3 IDENTIFICATION OF IMOS

The first thing that should be done is to segment the image, which will be required regardless of which techniques are applied afterwards. The segmentation is required to identify Independently Moving Objects (IMOs) and thereby establishing which parts of the image belongs to the object(s) we want to track.

The first thing that needs to be done is to divide all pixels into two classes: IMO and background [Chumerin08].

By looking at the parts of the image whose vectors start or end in the same vanishing point (also called the focus of expansion), then one can assume that these belong to the same object. In the same way, the vectors for the background (the world) should also end or start in a specific vanishing point, see Figure 33.

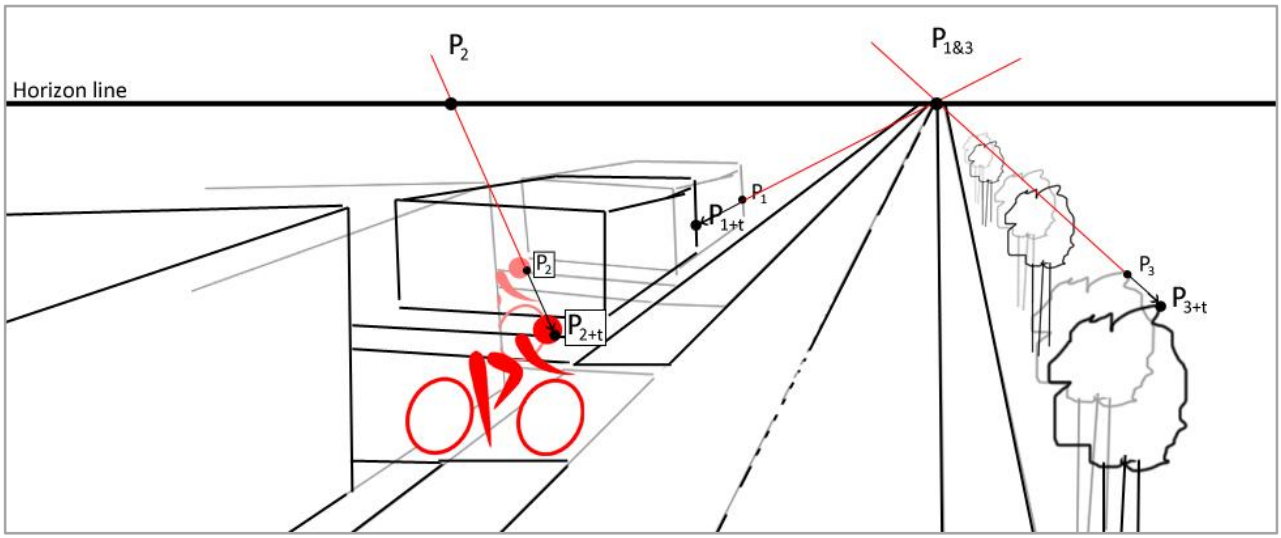


Figure 33: Showing different vanishing points for background and objects.

In order to find the vanishing points one need to first find the horizon line. This is the line at which there is no velocity on the y-axis.

The system could also benefit from segmentation by remembering the location in previous frames. This is also explained in the following sections.

Two methods, described below, used to find the horizon line will be implemented. If both work satisfactorily they will both be used, as this should provide a better estimation of the placement of the horizon line in motion field with noise.

Finding the horizon line

The first method makes use of the above mentioned definition that there is no velocity on the y-axis on the horizon line. The sum of all the pixels vertical velocities in every horizontal line of each frame will be calculated and the line with the lowest score will be used. The result will be improved if the ten lines with the lowest score will be used and if the system uses the results from earlier calculations to base the new calculation on.

The second method is to use the velocity field on objects that have the same movement in the image and calculate their intersection. If all velocity vectors come from the same object then all the vectors will intersect in the same point, see Figure 37. It will often be velocity information from the background which will be used to calculate the intersection point. The reason for this is that the image will consist of areas without IMOs. It is important to have at least two areas widely spaced in the image with no IMOs, as the calculation of the intersection from two vectors being placed close to each other will give a relative high error rate because a small error on the angle of the velocity vector will be highly influential.

The system needs to account for the possibility that the calculation in a single frame might not provide a usable result and in this case the result from the previous frames should therefore be used.

Segmentation from vanishing point

With the horizon line found, the vanishing point for a certain vector can be calculated by combining the position of it and its slope. This is a simplified approach inspired by [Pauwels04] which utilizes a method to segment IMOs in noisy optic flow fields.

There is however a problem when determining which vectors belong to the background and not to IMOs. Here a solution can assume that some regions in the image are more certain to have vectors belonging to the world.

Such regions can be the top and bottom right corners where there is little risk of being IMOs. The vectors in such regions can then be used to determine the vanishing point of the background and all vectors with the same vanishing points can be segmented away.

When having found the IMOs in the frame then one needs to segment away the IMOs that are noise. There will always be pixels not belonging to IMOs, i.e. vectors with the same vanishing point as the vectors on the IMO. The segmentation is done by doing neighborhood processing. If there are many neighbors with the same vanishing points then it is properly an IMO, as opposed to pixels were none or only very few neighboring pixel share intersection points.

Segmentation through memory

The system will also benefit from having incorporated memory of where the IMO is assumed to be. The memory is used to store the position of the expected IMO pixels. This position is compared with the position of a new IMO's pixels and if there have been any IMOs close to the position in earlier frames then it is more likely to be a pixel belonging to an IMO. The characteristic of noise disguised as an IMOs is that it will appear on a more or less accidental place in the image and disappear again immediately after. Memory in the system will be able to ignore these as IMOs.

The might be multiple IMOs in the frames, such as walking people, runners or multiple cyclists. As mentioned in "2.6 Success Criteria", this system is focused on cyclists but effort to avoid detection of pedestrians will not be done as one might as well want to include those also. In the case of multiple relevant IMOs, the system should, as mentioned in "3.5 Requirement Specification" find the one in most danger, based on speed and position, and then check if a collision is eminent for this one. This lead to the design of the system which has to tell if the IMO has a placement and velocity which puts it in a dangerous situation.

4.4.4.4 DECIDING IF IMO IS IN DANGER

The system will be designed to monitor the critical region from c_2 (see Figure 28) to somewhere on the side of the car. But if the system should be more than a simple motion detector it has to be able to compute if a cyclist in this region is really in danger or not. First it is needed to understand precise how the system has to work. The first question is when to activate the system. There are overall three degrees of activation:

- Active at all times.
- Active when the right turn indicator is turned on.
- Active only when turning.

Having the system active when the right turn indicator is turned on seems like a reasonable compromise as this should leave it active in all required situations without annoying the driver excessively. This was also how one of the projects in "2.2.1.1 Collisions Between Vehicles" successfully chose to activate their system. It should be noted that even though the system is not active and thereby not outputting any detection print outs, it might still be gathering data.

Activating the system with the right-turn indicator only gives the clue that a turn will be made within an unknown and arbitrary amount of time. The exact time until a turn is made, and thereby the exact time until a potential collision occurs, can only be calculated if the distance to the junction is known. This however is not something that is simple to find and it is certainly not within the scope of this project.

What can be used is the fact that the car potentially can make a turn at all times after the right turn indicator is put on. This means that the system has to compute the potential collision situation at all times while the indicator (and thus the system) is activated.

What the system has to do to determine if a detected cyclist is hit by the car that turns the millisecond after the calculation is to:

- Calculate the distance to the cyclist.
- Calculate the relative velocity of the cyclist.
- Get the velocity of the car as a input.
- Calculate if the cyclist is in the dynamic critical region.

As done earlier in this chapter the relations between the critical value of the distance to the cyclist, the car and cyclists velocities is investigated. In stead of finding the maximum distance to the cyclist (c_2) here the need is to use the ever changing values of the velocity of the car and cyclist to determine a critical distance $L_{cri}\ [m]$ from where the system have to detect the cyclist.

$$L_{cri} = \left(\frac{v_b}{v_c} - 1\right)(L_1 + L_c)$$

Where L_1 and L_c are vehicle dependent variables, L_{cri} is the distance to the cyclist and v_b and v_c is the velocity of the cyclist and the car respectively. It is possible to get the velocity of the car directly from its speedometer, or e.g. a GPS-receiver.

The only unknown variable in the equation is the velocity of the cyclist. This is the procedure to find it:

The only information about the cyclist's velocity is the magnitude of the horizontal optical flow velocity vectors from the pixels on the cyclist. The magnitudes of the vectors vary depending on the distance and angle to the observer. The relative velocity $v_{b.rel}\left[\frac{m}{s}\right]$ of the cyclist to the car can be calculated as

$$v_{b.rel} = \frac{l_f}{t_f}$$

where l_f [m] is the distance the cyclist travels in the time t_f [s]. As we get the velocity information between every frame, t_f is the time gap between two frames. With a rate of 25 frames per second the time elapsed is 1/25 sec. The magnitude of the velocity vector l_{pixel} is given in pixels, so it is needed to convert the length of the vectors in pixel in the image to a distance traveled in the real world. There the assumption about the cyclist placement 0.5 meter orthogonal from the car is used. Consider a camera with a angle of view of 50˚ pointing orthogonal out of the car is can be shown that the camera covers 0.46 m. If the width of the image is l_w the relationship is:

$$l_f = l_{pixel}\frac{l_w}{0.46m}\ (\ldots)$$

This is however not the only scale needed to be implemented. The magnitude of the velocity vectors in the image change with the distance and the angle from the viewer. Let us first consider the situation in Figure 34.

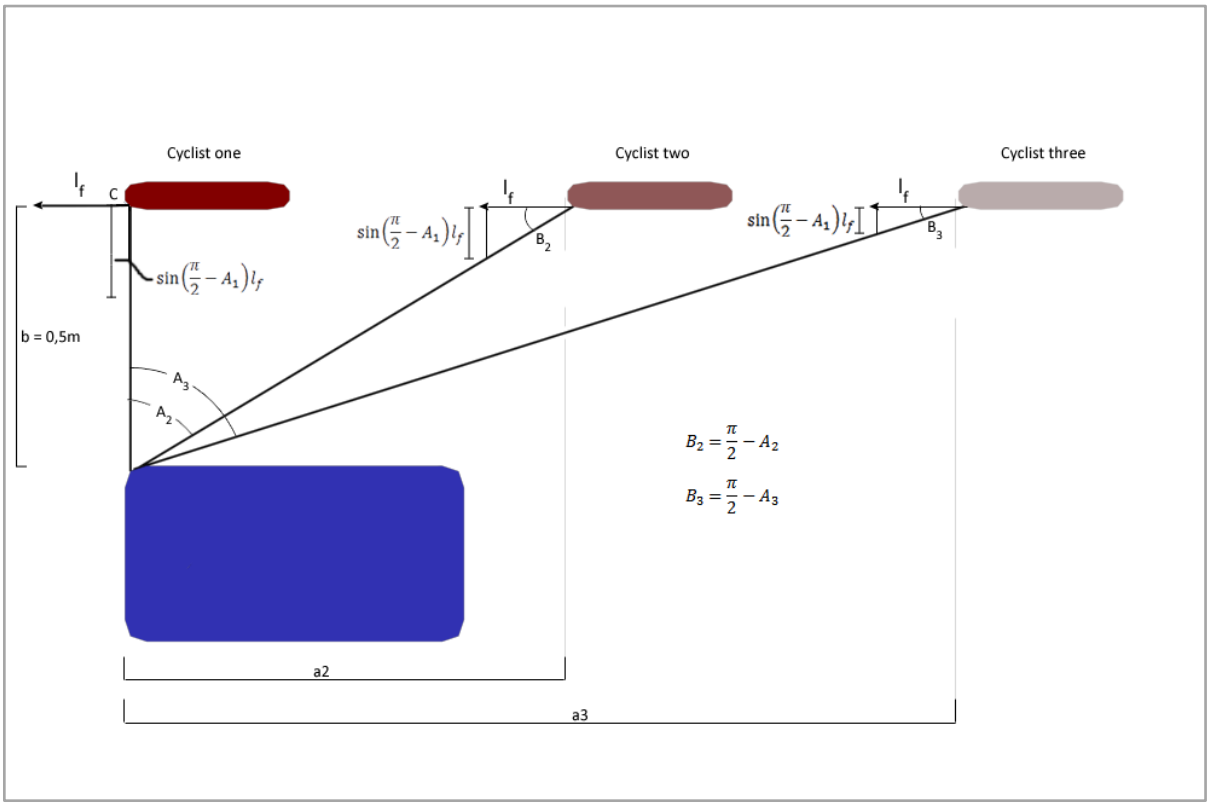


Figure 34: Relationship between angle and distance calculating vector magnitude.

If the angle *A* is 0˚ then the length traveled by the cyclist is calculated as in the above formula. But a less acute angel makes the magnitude of the vector smaller with the same velocity in the real world. If angle *A* is 90˚ then the magnitude of the vector will be infinitely small. From simple angle reflection it can be shown that the relation between the distance covered by the cyclist and the angle of view is:

$$l_f = l_{pixel}\frac{l_w}{0.46m}\frac{1}{\sin(\frac{\pi}{2}-A)}(\ldots)$$

where *A* is calculated with the cosine relations because all of the side in the triangle *ABC* is known (more on how to find the length *a* later).

It is a fact the size of an object in an image is cut in half if the distance between the camera and the object is doubled. This implies that we normalize the length of the vectors with the distance of 0.5 m which is the point of reference when calculating the pixel per meter relation. In total, the length the cyclist covers between two frames is

$$l_f = l_{pixel}\frac{l_w}{0.46m}\frac{1}{\sin(\frac{\pi}{2}-A)}\frac{\sqrt{0.5^2+L_{ac}}}{0.5}$$

Now the relative velocity of the cyclist is calculated as

$$v_{rel} = \frac{l_f}{t_f} = l_{pixel}\frac{l_w}{0.46m}\frac{1}{\sin(\frac{\pi}{2}-A)}\frac{\sqrt{0.5^2+L_{ac}}}{0.5}25$$

where L_{ac} is the actual placement of the cyclist. To get the velocity of the cyclist v_b:

$$v_{rel} = v_b - v_c \Leftrightarrow v_b = v_{rel} + v_c$$

This result is put into $L_{cri} = \left(\frac{v_b}{v_c} - 1\right)(L_1 + L_c)$ and is compared with the actual placement of the cyclist. This placement of the cyclist or the distance from the car is calculated solely by the cyclist's placement in the image. This is a simplified version of distance measurement, which includes some disadvantages, e.g. the measure problems occurring if the cyclist is moving towards or away from the side of the car. Nevertheless the system is now able to tell if a detected cyclist is in danger of being hit, if the car is turning right immediately after the computation. So if L_{ac} is the placement of the cyclist in the real world and $L_{ac} \leq L_{cri}$ then the system has to issue a detection output.

4.5 CONCLUSION

The goal of "4 Design" has been to fulfill the "3.5 Requirement Specification", specifically by going from what the product has to do and explaining how it will do it. Optical flow was chosen as the best approach for the system and then the two variants of optical flow, dense and sparse, were compared. Dense was ultimately chosen because it tries to compare motion for all pixels, as opposed to sparse methods which only looks for features. This means that there should be a significantly higher theoretical chance of finding vectors on the cyclist. Four dense optical flow methods were found, dense Lucas-Kanade, Horn & Schunck, Farnebäck and Block Matching, and these were compared to see which one would suit the final system best. Dense Lucas-Kanade was chosen as the first priority for implementation. These methods were partly chosen because they are already available in OpenCV.

Use Case 1, with a cyclist overtaking the car and therefore a camera pointing backwards, was chosen as the focus for the rest of the project and it was explained in exactly what direction the camera should point and how much it should cover. Then the design for the detection part was explained, which consists of a system which should make it possible to isolate and detect the cyclist. Specifically, it will try to segment the cyclist by looking at the horizon line and vanishing points for objects and the background in the video. A memory system will also be added to increase the quality of cyclist detections. Lastly, the system should be able to determine whether or not a cyclist is in danger depending on where the cyclist is in the video frame.

The following chapter will implement what was specified in "4 Design".

5. IMPLEMENTATION

Based on the knowledge from "4 Design", an implementation of the product will now follow. Since implementation is the design plans carried out, any deviations from the design will be specified.

During "4 Design" it was determined that four different dense optical flow algorithms should be implemented in order to see which one works better. The algorithms will be implemented in the order listed below. It should be noted that if an algorithm is successful the remaining will likely not be implemented. The four are as follows:

- Dense Lucas-Kanade
- Horn & Schunck
- Block Matching
- Farnebäck

If the results from one of the algorithms look promising "4.4.4 Design of the Detection Part" will be added to the algorithm. Specifically the detection part is used to perform segmentation, based on the data from the optical flow algorithm, to detect the cyclist.

If none of the four dense optical flow algorithms are successfully implemented, alternatives might be considered instead.

Before going into the technical details the development process and tools involved will be explained.

5.1 PROCESS

The solution is implemented in C++ as this language delivers the performance required.

The development of the program will take place in the Integrated Development Environment Microsoft Visual Studio (Xcode on Macs). The computer vision library OpenCV is used in order to gain easier access to image processing functions. When there is a built-in function in OpenCV that fits the task which needs to be done, the built-in function is chosen in favour of making a new and possibly slower one.

Subversion (SVN) is going to be used to ensure the coherence between and efficiency of multiple programmers working simultaneously on the same files.

In order to keep the code consistent it was agreed upon that camelCase should be used as naming scheme. This means that e.g. all functions names will start with a lower case letter and folloing words will be indicated by an upper case letter, like in the name camelCase.

5.2 FOOTAGE AND PREREQUISITES

The footage used to test each algorithm will be denoted as the Implementation Footage and was recorded simultaneously with the Test Footage, which will be used for "6 Test". The specifics of this recording session were therefore identical to that of the test footage explained in "6.1 Test Setup". The footage consists of 10 sequences for Use Case 1, and 10 for Use Case 4.

The Implementation Footage was recorded on a Canon MiniDV MD215 camcorder, which fulfils the criteria mentioned in "4.4.4.1 Prerequisites" by recording with 25 fps, having a decent image sensor and an angle of view with more than 50 degrees. As discovered in "4.4.3 Choosing an Use Case", the optimal placement of the camera is by mounting it on the foremost, right side of the bonnet, and having the edge of the field of view

being able to just see an object 41.53 meters behind the car and 0.5 meters orthogonally to the side of the car (see Figure 30 on page 58).

5.3 Optical Flow Algorithms

This section describes the implementation of the individual optical flow methods in the order defined in Table 4. Since dense Lucas-Kanade showed the most potential in "4.3 Choosing a Particular Dense Algorithm" it will be implemented first.

5.3.1 Dense Lucas-Kanade

The dense Lucas-Kanade algorithm is, as described in "4 Design", incorporated in OpenCV and should therefore be straightforward to implement. The code for the system is a modified version of [Stonerain08]. It starts by loading and decoding a video, then it will go through each frame of the video by using a while loop as seen here:

```
CvCapture *input_video = cvCaptureFromFile("video.avi");
number_of_frames = (int)cvGetCaptureProperty (input_video,
CV_CAP_PROP_POS_FRAMES);
long current_frame = 0;
while(current_frame <= number_of_frames )
{
     cvSetCaptureProperty(input_video, CV_CAP_PROP_POS_FRAMES,
     current_frame);
     IplImage * frame = cvQueryFrame(input_video);
     cvConvertImage(frame, image[0], CV_CVTIMG_FLIP);
     ++current_frame;
     cvSetCaptureProperty(input_video, CV_CAP_PROP_POS_FRAMES,
     current_frame);
     frame = cvQueryFrame(input_video);
}
cvConvertImage(frame, image[1], CV_CVTIMG_FLIP);
```

Code Block 6: Playing a video with a while-loop.

The code snippet loads a decoded frame into *image[0]*, then moves to the following frame and loads that frame into *image[1]*. By doing this *image[1]* will always be one frame ahead of *image[0]*. If the executable generated by this code is running on the Windows operating system, the frames will be upside down due to a bug in OpenCV, so the *CV_CVTIMG_FLIP* flag will flip the frames.

```
cvCvtColor (image[ 0 ], source[ 0 ], CV_BGR2GRAY);
cvCvtColor (image[ 1 ], source[ 1 ], CV_BGR2GRAY);
IplImage    * velocityX = cvCreateImage(size, IPL_DEPTH_32F, 1),
            * velocityY = cvCreateImage(size, IPL_DEPTH_32F, 1);
```

Code Block 7: Converting images to grayscale and creating image holders.

In Code Block 7 the two frames, *image[0]* and *image[1]* are converted to gray and stored as *source[0]* and *source[1]* followed by two 32bit float images which will be created as two image holders called *velocityX* and *velocityY* which can store both positive and negative values. These two images will be used to store the horizontal and vertical movements from one frame to the other when using the dense Lucas-Kanade algorithm:

```
cvCalcOpticalFlowLK(source[0], source[1], cvSize(1,1),
velocityX, velocityY);
```

Code Block 8: Calculating optical flow with the Lucas-Kanade algorithm.

source[0] is the first frame and *source[1]* is the following frame. The difference in pixel positions in these two images will be calculated by the Lucas-Kanade dense algorithm. The changes to the horizontal direction will be stored in the image *velocityX* and the changes in the vertical direction will be stored in *velocityY*. *cvSize()* is the window size used to determine the size of the groups of pixels the algorithm should operate on and thereby the level of detail. Lower values make it approach a dense level while higher values make it more sparse. Here a window size of (1,1) is used making it as dense as it can get. Other values for cvSize() were also tested but with no improvement in the output. The method is tested on footage of a cyclist overtaking a car as seen below.

Figure 35: Original Implementation Footage.

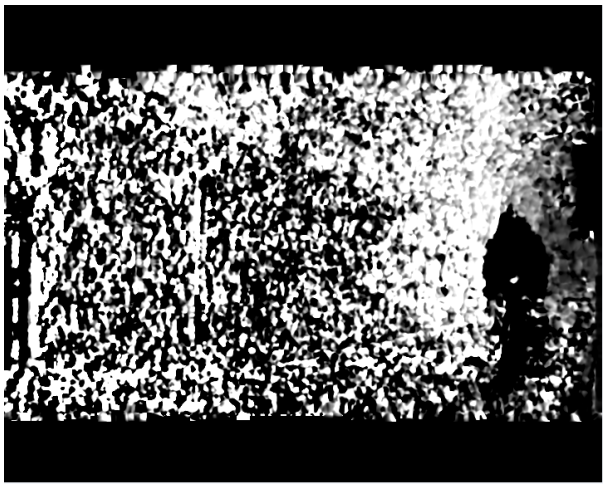


Figure 36: Running dense Lucas-Kanade on Implementation Footage.

Looking at Figure 36 above it is apparent the cyclist clearly stands out from the background. It has to be noted that the black sections in the image where the cyclist is contains no velocity information. In other words: The algorithm finds plenty of motion in the background but none on the cyclist.

At this point it would make sense to perform Dilation and Erosion to clearly segment the cyclist from the background. This, however, would not work because once the cyclist gets closer he goes from black to being covered in white. From this we have to conclude that the algorithm provides too inconsistent results to be used for this project, which means that work has to proceed to the next algorithm: Horn & Schunck.

5.3.2 HORN & SCHUNCK

The Horn & Schunck method is the second of the four chosen dense optical flow methods to be implemented. It was one of the first of these kinds of techniques to make use of brightness constancy assumption, as explained in "3.3.5.3 Optical Flow", and it relies on iterations to solve its equations, all of which is also the case for Lucas-Kanade. The code used is found at [Flair07]. The key parts of the implemented code will be described in this section. The code to load in video and split it in a current and previous frame is similar to the code used for the dense Lukas-Kanade.

The function that computes the Horn & Schunck algorithm is *cvCalcOpticalFlowHS();* and its prototype is:

```
cvCalcOpticalFlowHS(
     const CvArr* prev,
     const CvArr* curr,
     int use_previous,
     CvArr* velx,
     CvArr vely,
     double lamda,
     CvTermCriteria criteria );
```

Code Block 9: Prototype for Horn & Schunck algorithm in OpenCV.

The function includes a previous and a current image as the main input. Both are 8-bit single channel images. The *usePrevious* parameter uses data from previous frame as the initial starting point. The *CvArr velx* and *CvArr vely* is velocity in x and y directions stored in 32 bit floating point images. Double *lamda* 1 is a parameter which is weight related to the Lagrange multipliers which represents the relative weight given to errors as one attempts to minimize equations related to motion-brightness and smoothness [Bradski08]. The *CvTermCriteria* criteria determines the amount of iteration of the Horn & Schunck process on one frame. Below are the inputs used for the implementation of the method.

```
cvCalcOpticalFlowHS(gray_img, prev_img,
usePrevious, velx, vely, 1, IterCriteria);
```

Code Block 10: Calculating optical flow with the Horn & Schunk method.

This yielded the following results:

Figure 37: Original Implementation Footage.

Figure 38: Running Horn & Schunck on Implementation Footage.

In Figure 38 above, the Horn & Schunck method is tested on the same implementation footage as with the Lucas-Kanade. In this image the cyclist contains white pixels which include information of directions. When looking at the pixel values for a cyclist, we see information, but segmenting away all pixels translating right-to-left, then also the cyclist's pixels are removed even though these should remain. This error appears to have its roots in the inaccuracy of the Horn & Schunck method.

Segmenting the cyclist was therefore not a success for the Horn & Schunck method and was then put on hold. The work proceeds with the next algorithm: Block matching.

5.3.3 BLOCK MATCHING

Block Matching is the third in line to be implemented. It encompasses a lot of similar algorithms in which the image is divided into small regions called *blocks*. These algorithms try to divide the previous and current images into such blocks and then find the motion of these blocks [Bradski08]. The code used was found at [Stonerain08].

The function for the block matching algorithm in OpenCV is called *cvCalcOpticalFlowBM()* and its prototype can be seen in Code Block 11.

```
void cvCalcOpticalFlowBM(
     const CvArr* prev,
     const CvArr* curr,
     CvSize blockSize,
     CvSize shiftSize,
     CvSize max_range,
     int usePrevious,
     CvArr* velx,
     CvArr* vely)
```

Code Block 11: Prototype for the Block Matching algorithm in OpenCV.

As with the other optical flow functions there is no direct output of the function and instead there is a parameter output through the last two arguments of the function, the *velx* and *vely* that is. As with the previous two optical flow methods it takes the previous frame and the current frame as input (both 8-bit, single-channel images), which are the two first arguments for the function.

The *cvSize()* input is the third argument which determines the block size which corresponds roughly to the window size as described for the Lucas-Kanade algorithm. The second *cvSize()* input determines at which point it should start considering a new block. Depending on how this relates to the block size this makes it possible to have overlapping blocks.

The *max_range* input determines how far around each block it has to search in order to find the block in its new position. If *use_previous* is changed from 0 to 1 it will override the *max_range* and instead use the velocities found from a previous function call in order to determine how far to search. The function was run with the following parameters:

```
cvCalcOpticalFlowBM(source[0], source[1], cvSize(16, 16),
cvSize(1, 1), cvSize(11, 11), 0, velocityX, velocityY);
```

Code Block 12: Running Block Matching function.

This yielded the following results:

Figure 39: Original Implementation Footage.

Figure 40: Running Block Matching on Implementation Footage.

The implementation footage is once again used and as with the dense Lucas-Kanade method there is only information from the background when the cyclist is far away. The issue is again that this is not consistent since there is information on the cyclist when he gets close. This inconsistency is, just as with the dense Lucas-Kanade, a significant issue in itself. In addition to that, the velocity information from the cyclist when up close is highly chaotic, similarly to the results seen from Horn & Schunck.

The Block Matching algorithm thereby seems to have the issues from both dense Lucas-Kanade and Horn & Schunck meaning that it is ultimately the weakest of the three. Implementation therefore has to proceed to the last of the four planned algorithms, namely Gunnar Färneback.

5.3.4 FARNEBÄCK

As the last algorithm from the implementation plan the Farnebäck algorithm [Farneback02] was also implemented. This algorithm required OpenCV 2.0 which therefore had to be installed.

The key part of the implementation code is the call to *calcOpticalFlowFarneback()* seen below.

```
void calcOpticalFlowFarneback(
     const Mat& prevImg,
     const Mat& nextImg, Mat& flow,
     double pyrScale,
     int levels,
     int winsize,
     int iterations,
     int polyN,
     double polySigma,
     int flags);
```

Code Block 13: Prototype for the dense Farnebåck algorithm.

Since this algorithm was only recently added to OpenCV, in May of 2009, there is not a lot of useful examples using it. The official OpenCV reference manual does however explain all the inputs [OpenCV09]: *prevImg* and *nextImg* denote the two input images, both 8-bit single-channel. *flow* is the computed flow image,

pyrScale specifies the image scale, always less than 1, to build the pyramids for each image. So with a value of 0.5 each next layer is twice as small as the previous. *levels* denote the number of pyramid layers including the initial image, *winsize* is the average window size, larger values increase the algorithm's robustness to image noise and increase the chances for fast motion detection, but yield a more blurry motion field. *iterations* denote the amount of iterations for each pyramid level, polyN is the size of the pixel neighborhood used to find expansions in each pixel. The larger values give more robustness but a more blurry motion field. *polySigma* is a standard deviation used to smooth derivatives that are used as a basis for the polynomial expansion[8], and *polySigma* should follow the values of *polyN*. Lastly *flags* denote the operation flags, and can either be *OPTFLOW_FARNEBACK_GAUSSIAN*, which usually gives more accurate flow at the cost of speed, or *OPTFLOW_USE_INITIAL_FLOW*. The specific parameters used for our implementation will be explained in Code Block 17 below, as a starting point parameters found in [Damien09] was used, and they yielded the result seen in Figure 41.

Compared to the other optical flow algorithms Farnebäck takes the two frames as matrice input, which means the *IplImages* have to be converted to 8bit MxN matrices first. The code for this is seen below:

```
cv::Mat im1 = cv::cvarrToMat(source[ 0 ]),
im2 = cv::cvarrToMat(source[ 1 ]);
```

Code Block 14: IplImages *converted to matrices in Farnebäck.*

The *flow0* is the output in 32 bit format which contains the flow values at F(x,y). In order to work with the flow values it was needed to separate them which is done in this code:

```
for( int y = 0; y < flow->height; y++ )
{
     float* vx = (float*)(velx->imageData + velx->widthStep*y);
     float* vy = (float*)(vely->imageData + vely->widthStep*y);
     const float* f = (const float*)(flow->imageData + flow->widthStep*y);
     for( int x = 0; x < flow->width; x++ )
     {
             vx[x] = f[2*x];
             vy[x] = f[2*x+1];
     }
}
```

Code Block 15: Separation of flow values in Farnebäck.

With the flow values separated following is then a large chunk of code which calculates which pixels the above mentioned flow values correspond to in the original picture. At this point the output looked like this:

[8] Raises an equation to the *n*th power.

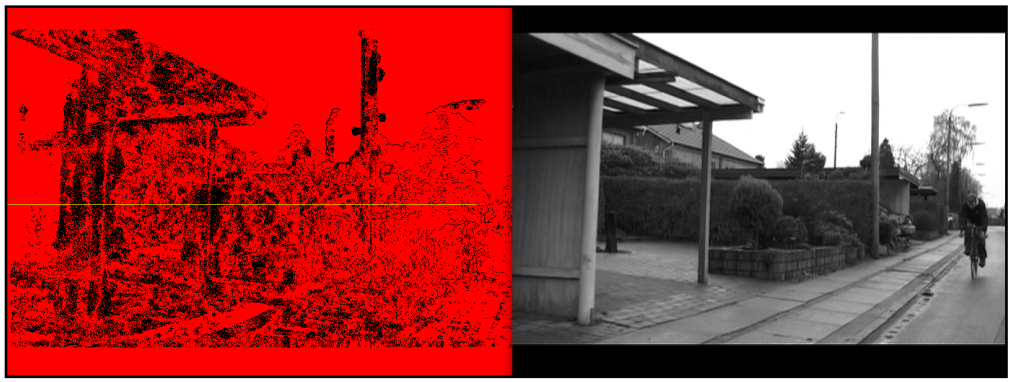


Figure 41: Running Farnebäck for the first time.

All we want to track is the cyclist, but right now the algorithm tracks pixels in most of the image. To remove noise, a simple segmentation was done. Since all coordinates for the moving points are known, it was possible to remove all points that were moving in the opposite direction of the cyclist, i.e. to the right. This was done with a simple conditional structure that only draws the lines moving to the left:

```
if ((p2.x < p.x && p2.y < p.y) || (p2.x < p.x && p2.y > p.y))
{
     cvLine(vel, p2, p2, CV_RGB(255,0,0), 1, 8);
}
```

Code Block 16: Segmenting away noise moving right.

This gave the following output:

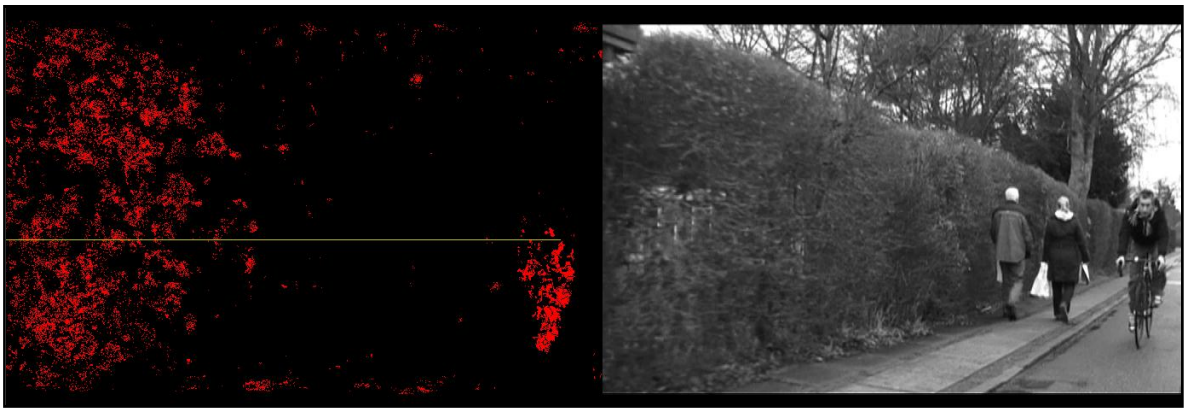


Figure 42: Running Farnebäck after removing noise, which is moving to the right.

Here it is quite obvious where the cyclist is. However, there is still a significant amount of noise in the left side of the image, and the amount of detected pixels on the cyclist quickly degraded when the cyclist was only a bit longer away. However, we had still not tried to tweak the output of the algorithm by changing the input parameters, so this was the next step.

As mentioned, the following parameters were taken from [Damien09], and was used a foundation for experimentation.

```
calcOpticalFlowFarneback( im1, im2, _flow, 0.5, 2, 10, 2, 14, 2.0,
cv::OPTFLOW_FARNEBACK_GAUSSIAN);
```

Code Block 17: Initial inputs for the Farnebäck algorithm.

These parameters yielded the result in Figure 42. At this point, the removal of the noise on the left was a big priority, so the number of pyramid layers were increased from 2 to 4. This gave a significant improvement, see Figure 43.

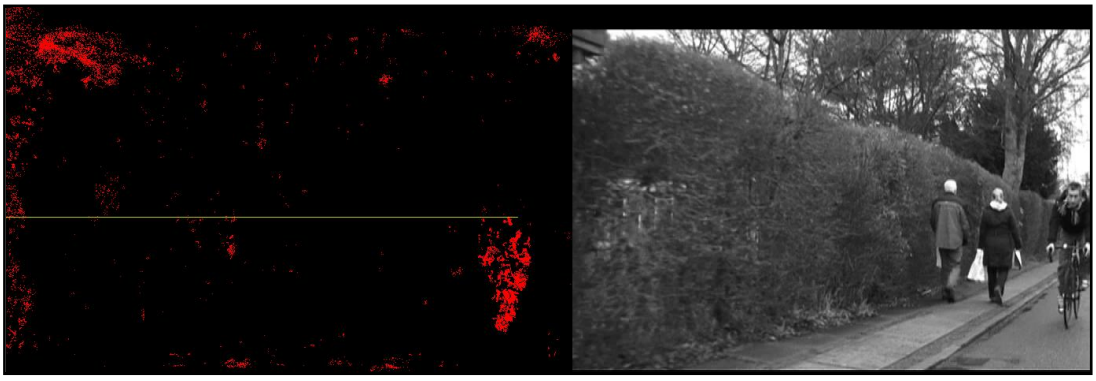


Figure 43: Running Farnebäck after increasing amount of pyramids.

Changing the parameters for the amount of iterations, the size of the pixel neighborhood or *polySigma* yielded no significant changes. At this point the algorithm seemed to be working well enough to start improving it by adding the detection part from "4.4.4 Design of the Detection Part". More specifically the segmentation removing the pixel vectors which intersect on a specific range on the horizon line to the vanishing point.

5.3.4.1 SEGMENTATION OF THE CYCLIST

In "4.4.4.3 Identification of IMOs" two methods for estimating the horizon line were described. The first method estimates the vanishing point by looking at vectors in areas of the frame which is certain to be background. The second method estimates the vanishing point by looking at all vectors in the frame. Since the second method looks at all vectors it should per default give a more trustworthy result, and therefore it is used for this project.

The horizon line to the vanishing point was approximated based on the flow pixels with the least frame-to-frame movement on the y-axis as seen in Code Block 18:

```
if (abs(p2.x - p.x) != 0)
{
     totalY += abs(p2.y - p.y);      // Collecting changes in y-values
     timesToDivide++;
}

// Calculating the intersection points for a line.
double angle;      angle = atan2( (double) dy, (double) dx );
double angle2;     angle2 = abs(angle);
double hypotenuse; hypotenuse = sqrt( pow(dy, 2) + pow(dx, 2) );
double slope;      slope = (dy)/(dx+0.00000000000000000001);
double b;          b = dy-(slope*dx);                    // intersection

intersection = (finalPlacering - b) / (slope+0.00000000000000000001);
```

Code Block 18: Calculation of intersection points on the horizon line to the vanishing point.

And the following code looks at the collected changes in y-values and finds the smallest change, which is most likely the horizon line to the vanishing point.

```
for (int i = 1; i < yVector.size(); i++)
{
    if (temp < yVector[i])
    {
        temp = yVector[i];
        placering = i;
    }
}
```

Code Block 19: Finding smallest change in y-values.

Unfortunately, this method did not prove to be an improvement, because it did in fact not only remove noise, but also too many of the detected features on the cyclist. It is believed this is more so due to limitations in our implementation than due to weaknesses in the algorithm itself.

Instead it was then decided to use erosion to remove the noise, since the detected features of the cyclist looked more dense than the noise, i.e. there was more black spots in the noise. Erosion and dilation steps were executed as seen in Code Block 20 (explaining the code for erosion) and Figure 44 below. Through trial and error it was found that a rectangle-shaped kernel with a size of 6 pixels was useful as this eliminated all noise and still left the cyclist in a lot of frames.

```
int rows = 6;
int columns = 6;
myKernel = cvCreateStructuringElementEx(rows, columns,
cvFloor(rows / 2), cvFloor(columns / 2), CV_SHAPE_RECT, NULL);
cvErode(vel, vel, myKernel, 1);
```

Code Block 20: Creating the kernel for erosion and executing erosion.

This gave the following result:

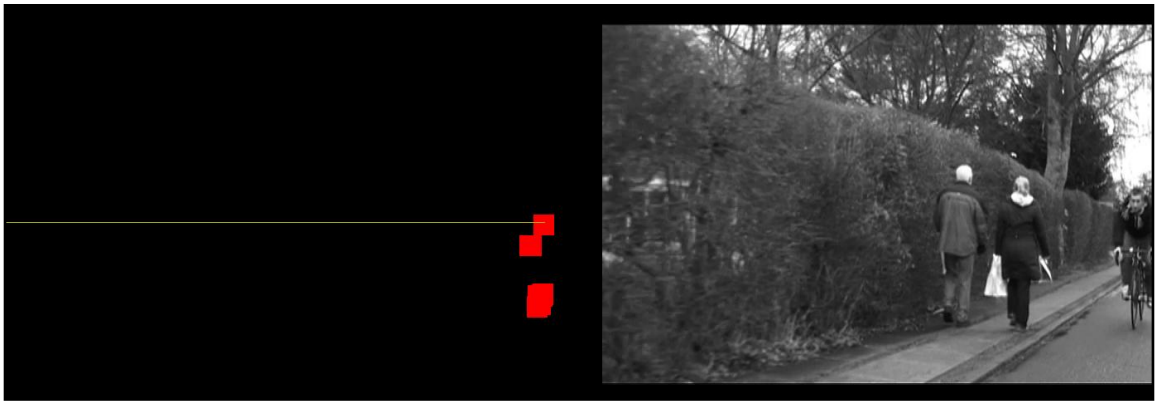


Figure 44: Running Farnebäck after performing Erosion and Dilation.

Even though the detection seems successful in Figure 44, the algorithm had a hard time detecting the cyclist in all frames, and there were still some noisy BLOBs. As mentioned in "4.4.4.3 Identification of IMOs" the detection part would benefit from incorporating a memory system. This system remembers BLOBs depending on how far away they were from BLOBs in previous frames. This algorithm pushed the coordinates from each BLOB into a vector, and then compared the BLOBs in the newest frame with earlier BLOBs, before determining whether or not the BLOB was a cyclist. For example, the algorithm could be set to look within a rectangle of

200 x 200 pixels in the last 5 frames. So if the newest BLOB had the coordinates (100, 100) and a BLOB two frames earlier had the coordinates (150,150) the algorithm would guess this was a BLOB for the cyclist. This improved the correct detection of the cyclist.

According to "4.4.4 Design of the Detection Part" the next step would be to determine whether the detected cyclist is in danger. However, even though the Farnebäck algorithm worked fine at a certain distance, it was concluded that it was not good enough to detect the cyclist within the necessary distance. First tests of Farnebäck also showed sluggish performance with each frame taking around a second to calculate. Therefore the Farnebäck algorithm had to be discarded. Since Farnebäck was the last of the four dense optical flow methods found in "4.3 Choosing a Particular Dense Algorithm" we were now forced to look elsewhere.

It seemed logical to take another look at the sparse Lucas-Kanade method used in the "3.4.2 The Simple Dynamic System".

5.3.5 SPARSE LUCAS-KANADE

Having three dense algorithms before it yield unsatisfying results, it did not seem wise to rely solely on Farnebäck. So, alongside implementing Farnebäck, work was commenced to get the sparse Lucas-Kanade from the Analysis Test up and running with proper segmentation as a backup plan.

The thought of a possible comparative test between Farnebäck and sparse Lucas-Kanade in case both worked out also seemed intriguing, but this was obviously not possible since the Farnebäck method had to be discarded.

This meant that the code from the Analysis Test would be used as a base, and several parts of existing code from the implementation of the four dense optical flow algorithms could be ported over. The basic principles of the method is explained in "3.4.2 The Simple Dynamic System", and the following sections will explain the implementation of the detection part (segmentation of the cyclist) from "4.4.4 Design of the Detection Part".

The sparse version of the Lucas-Kanade algorithm is similar to the dense algorithm described earlier on but differs in the way that it is based on features and thereby considered sparse. In OpenCV the features are determined with the cv*GoodFeaturesToTrack()* function which is based on the Shi-Tomasi method as explained in "3.3.5.3 Optical Flow" and "3.4.2 The Simple Dynamic System".

5.3.5.1 SEGMENTATION OF CYCLIST

This first part focuses on doing segmentation by using the vanishing points for objects in the image frame, as explained in "4.4.4.3 Identification of IMOs". This proved unsuccessful when implementing Farnebäck (see 5.3.4 Farnebäck), but since this is a significantly different optical flow method (sparse as opposed to dense) it is reasonable to believe it should work. Each feature is segmented if they fit into three parameters:

- Intersection point with horizon line
- Length (distance between feature position in each frame)
- Slope

This means that feature vectors which intersect at a certain point on the horizon line, have a certain length, and a certain slope will either be ignored or kept.

Code-wise this is stated with the following line:

```
if ( intersection>400 && intersection<1200 &&
     hypotenuse>1.4 && slope<0.8 ) {...}
```

Code Block 21: Segmentation of cyclist.

Attempts were made to calculate the horizon line based on the method described in "4.4.4.3 Identification of IMOs" and in "5.3.4 Farnebäck", where one tries to find the pixel height where there is no vertical flow. This however did not prove successful, mainly due to the lack of features on and around the horizon which made predictions inaccurate. In the end it was found that a hardcoded value worked the best. This means that the current implementation has limitations if one e.g. is driving up a hill.

If a feature passes the three segmentation tests described above then its positions are added to an STL[9] vector which works like a dynamic array. This is the beginning of the implementation of the memory system described in "4.4.4.3 Identification of IMOs".

The above mentioned segmentation criteria turned out to do a decent job of segmenting the right features but there was still a significant amount of noise. In "4.4.4.3 Identification of IMOs" it was found that it should be possible to remove noise through use of the spatial and temporal properties of the features. Noise tends to be isolated, both spatially and temporally, which is a property that can be used for further segmentation. This code, as seen in Code Block 22, simply checks how many other nearby good features there are, either in the current frame or in previous frames.

The code has a number of limitations though, mainly that one can not distinguish (and thereby tweak) between features in the same frame or in previous frames. Also, the previous frames can potentially stretch several seconds back and still be valued equally much, instead of correctly being weighted less.

```
for (int j = 0; j<min(15,featureX.size()); j++)
{
     if (featureX[featureX.size()-1] <=
     featureX[featureX.size()-1-j] + maxDistance
     && featureX[featureX.size()-1]
     >= featureX[featureX.size()-1-j] - maxDistance
     && featureY[featureY.size()-1]
     <= featureY[featureY.size()-1-j] + maxDistance
     && featureY[featureY.size()-1]
     >= featureY[featureY.size()-1-j] - maxDistance)
     {
             found++;
     }
}
```

Code Block 22: Looking in a certain area in the current and previous frames for high quality features.

The count variable from the code above goes from 0-15 and is used directly as a measurement of the quality (i.e. the likelihood of this being a cyclist) of this feature. Per frame the feature with the highest quality is found and this one is then considered as frame quality and used for the later calculations. Looking back, a solution with the average quality of features could potentially have given better results.

With the frame quality in place a smoothing function is needed to smooth out the result based on the quality of the previous frames and thereby further reduce noise. Without this one a single mistaken frame could generate a quality large enough to give a false positive. Code Block 23 simply calculates the average quality of the last 30 frames.

[9] Standard Template Library.

```
for (int j = 0; j<min(30,bestFeatureQual.size()); j++)
{
    qualTotal += bestFeatureQual[bestFeatureQual.size()-1-j];
}
qualTotal = (qualTotal/min(30,bestFeatureQual.size()));
```

Code Block 23: Calculating average quality of previous 30 frames.

If the *qualTotal* above is above a certain threshold then that should mean that there is a cyclist in the current frame.

It was soon clear that a major issue of the first builds was the fact that frames with trees in caused most of the features to be found on the top of those trees as the *cvGoodFeaturesToTrack()* function apparently mainly found good features there. Looking at the sequences it was clear that the y-position of those trees was always higher than the cyclist which meant that ROI could provide a benefit to the tracking, as mentioned in "4.4.4 Design of the Detection Part".

At first the ROI was attempted implemented as early as possible in the code as this should obviously be the most optimal. The attempt was done with the OpenCV function *cvSetImageROI()* but after numerous trial and error phases it still did not work. The issue was that cropping had to be performed on a copied image and this new image did for unknown reasons not get accepted as input for Lucas-Kanade even though color depth, etc. were similar.

Instead a very dirty hack was made, as seen in Code Block 24, which was to simply ignore the features if they were below a certain y-value.

```
if (frame2_features[i].y <275)    continue;
```

Code Block 24: Hard coding ROI.

Adding ROI could potentially have given a major performance boost and also allowed for many more features on the cyclist. The final implementation has none of these benefits but was only chosen as a very last option as the right way did not prove to work, and there was no time left for further attempts at getting it to work.

The last section of the detection part, "4.4.4.4 Deciding if IMO is in Danger", describes that the system should be determined whether or not the cyclist is in danger. This is done by estimating and calculating the relative position and speed of the cyclist, and then using this information to calculate the part of the frame where the cyclist is in danger. However, at this point it is clear that this is too time consuming to be finished within the scope of this project. Instead it is necessary to hard code the values for this region. At this point there are no more planned features to implement, so we proceed to tweaking and optimizing the system.

5.3.5.2 TWEAKING AND OPTIMIZATION

With all the above mentioned features then the system worked decently and test code was implemented to output the percent of frames in which it believed there was a cyclist. This detection is based on the frame quality variable mentioned in "5.3.5.1 Segmentation of Cyclist", the threshold of which is set to 6 (i.e. if a quality is above 6, it is determined to be a cyclist). The first iteration was run on the first sequence from the implementation footage (mentioned in "5.2 Footage and Prerequisites") for Use Case 1 (with cyclist) and Use Case 4 (without cyclist).

- Use Case 1, sequence 1: 87%

- Use Case 4, sequence 1: 41%

An attempt was then done to improve those results. The three segmentation criteria (intersection, length and slope) were considered the most important and these were therefore tweaked. First we try to cut down on the 41% False Positives from the above results. After several attempts we came to the following results:

- Use Case 1, sequence 1: 83%
- Use Case 4, sequence 1: 14%

It was at this point clear that trying to tweak the program without further data would be too hard a task. Based on inspiration from training systems such as neural networks, a function for retrieving data for features on the cyclist was then implemented. This was done by manually defining a rectangle which encapsulated the cyclist and moved with it, which meant that the properties of all features within that rectangle could then be outputted. Looking at this data it was clear that the actual properties of the cyclist features were somewhat different from what had been used as segmentation values.

Based on the new data a new set of segmentation values were determined which gave significantly better results as seen below:

- Use Case 1, sequence 1: 90%
- Use Case 4, sequence 1: 0%

These data were considered to be pretty good and it was therefore determined to also test the code with other sequences. The result of the percent frames detected as containing a cyclist can be seen below:

- Use Case 1, sequence 2: 84%
- Use Case 4, sequence 2: 0%

The True Positives (correctly detected cyclists) dropped slightly, which was expected as the code was tweaked for another sequence. It was also noted that the False Positives (falsely detecting a cyclist) were still at 0% which meant that the threshold between believing there is a cyclist or not could be raised. With that one raised, tests were done on three sequences from each use case, results below:

- Use Case 1, sequence 1: 98%
- Use Case 1, sequence 2: 98%
- Use Case 1, sequence 7: 100%
- Use Case 4, sequence 1: 0%
- Use Case 4, sequence 2: 0%
- Use Case 4, sequence 7: 77%

These results were quite interesting. The True Positives were equally great even for a sequence which was significantly different from the two first. On the other hand, the third test sequence from Use Case 4 showed a value of 77% which was way off the expected value and shows just how hard a task it is to tweak such a system. Taking a closer look at that clip it was clear that it was some water puddles in that clip which somehow had the same feature properties as a cyclist. As the properties were very much the same as a cyclist, potentially because of the reflection or refraction, this meant that it was probably a fault of the Lucas-Kanade implementation and not anything that could be improved through tweaking of the segmentation algorithm.

Next up tests were done with tweaking other values in the program. E.g. the amount of frames it looked back for smoothing (i.e. average of values across multiple frames) was both tried increased and decreased, none of which was able to improve the final result. Other more low level tweaks such as increasing the amount of

features for Shi-Tomasi or the amount of pyramid levels for Lucas-Kanade were also tried without any improvement.

The system was at this point deemed ready for test as the results were satisfactory. In order to get the results out of the program a file writer function had to be made. At first a solution was tried using *OFStream* which almost seemed to work. It was discovered that this overwrote the first parts of the file after having outputted for some minutes though, so this solution had to be reworked. In the end we had to switch to an *fprintf* solution instead.

Profiling

Up until this point only little focus had been put on performance and optimization as it was deemed more important to make it work and make it work right before making it work fast. In order to be able to optimize the program profiling was needed. This was achieved using the standard C++ function *clock()* which has a few percent inaccuracy, but not enough to conflict with the accuracy need for some simple profiling. The numbers below are measured on Pentium 3 1.13 GHz CPU.

Using profiling it was found that each frame on average took 306 ms with the main offenders being *cvGoodFeaturesToTrack()* at 120 ms, the loop running through all features at 80 ms and *cvCalcOpticalFlowPyrLK()* at 20 ms. While one usually should optimize the most demanding function it was not possible in this case as the input for *cvGoodFeaturesToTrack()* was locked down and its internals were most likely highly optimized already.

It did not seem relevant to perform per-line optimizations and instead things were optimized on a structural level by moving as much code as possible from the feature loop to the frame loop and from the frame loop to the main block. This brought the frame time to 274ms.

A lot of visual debugging information was not needed for the final test and this was therefore stripped out using a *#define* preprocessor allowing these parts to be stripped out at compile time. This brought the frame time to exactly 200 ms which was considered as the best we could get it to without too much work. The final program was also tested on a faster 2.4 GHz dual core CPU which proved to be able to handle each frame in 44 ms on average. This means that with the frame rate of 25 fps it is almost running in real-time, as this requires a computation speed at less than 40 ms.

5.4 CONCLUSION

In this chapter four dense optical flow algorithms were implemented. Of the four dense optical flow methods the Farnebäck algorithm gave best results, but was eventually discarded due to detection issues when the cyclist was far away. Instead the sparse Lucas-Kanade method was looked at again, and it proved to show more promise of either of the dense methods. The detection part in "4.4.4 Design of the Detection Part" was implemented on the sparse Lucas-Kanade method, this included segmenting based on the horizon line, using a memory system to better determine the quality of vectors in the frame and hard coding region values of the frame in which a cyclist would be in danger.

The sparse Lucas-Kanade method was tweaked and optimized with the implementation footage. The results were quite positive and the system was deemed ready for testing, which will be done in the next chapter.

6. TEST

This chapter's primary purpose is to test if the Success Criteria (see "2.6 Success Criteria") are met. If they are met it will most likely be concluded that the project is a success, because the final problem statement has not only been answered, but answered upon with a satisfactory answer. More about this in "8 Conclusion".

No matter if the test ends in a negative or positive result a relevant question to answer is why it ended as it did. The answer to this is this chapter's secondary purpose. Overall, the results presented should show two things:

- If the Success Criteria are fulfilled.
- How well balanced the current system is, and how it can be improved upon.

The chapter is divided in three parts. First a presentation of the test setup, then a presentation of the results from the test, and last a discussion on what can be concluded from the results.

6.1 TEST SETUP

In "3.5.3.2 Functional Requirements" a total of four use cases about the cyclist's motion in relation to the car were created to break down the system into smaller parts. Due to delimitations done in "4.4.3 Choosing an Use Case" two use cases are tested in the chapter:

Use Case 1

Scenario: A cyclist is catching up to a car approaching a right turn situation in a cross section. In this case the front wheel of the bicycle is the first thing to enter system's field of view.

Use Case 4

Scenario: No cyclist is in the system's field of view.

The two use cases are placed in the columns of Table 6, with the two possible outcomes of the test in the rows. This gives a total of four outcomes from testing if the Success Criteria are met. In Use Case 1 when the cyclist is in view the system should detect him in more than 9 out of 10 situations (True Positives (TP)), and when no cyclist is in view the system should not detect a cyclist in less than 1 out of 10 situations (False Positives (FP)). This automatically means that there should be less than 1 out of 10 situations with False Negatives, and more than 9 out of 10 situations with True Negatives, e.g. the sum of the outcome in each use case must be 1.

	Bicycle in view (Use Case 1)	Bicycle not in view (Use Case 4)
Bicycle detected	True Positive (TP)	False Positive (FP)
Bicycle not detected	False Negative (FN)	True Negative (TN)

Table 6: Test scenarios.

The Final Test of the system should be considered to be both the Final Test and the Accept Test, since the requirements are the same as the Success Criteria for the entire project.

6.1.1 STATISTICS OF THE TEST

This section will describe how to conclude if the success criteria in Table 7 are met. The statistical area to examine is the field of inferences concerning proportions. In this field is possible to make estimations of

proportions, determine the sample size with a certain level of significance, determine the maximum error of the estimate of the proportion, and conduct hypotheses test concerning the size of one proportion.

	Bicycle in view (Use Case 1)	Bicycle not in view (Use Case 4)
Bicycle detected	Success more than 9 out of 10	Failure more than 1 out of 10
Bicycle not detected	Failure more than 1 out of 10	Success more than 9 out of 10

Table 7: Success Criteria.

Two independent proportions needs to be tested, one for each use case. In Use Case 1 it is desired to prove that the system detects more than 9 out of 10 cyclists, and in Use Case 4 it is desired to prove that the system detects less than 1 out of 10 cyclists. That is the reason why the test needs to run as two independent tests.

The information that is usually available when estimating (proving) a proportion is the number of times *X* an event occurs in *n* trials or observations. The point estimator of the true population proportion is the sample proportion $\frac{X}{n}$, referred to as the proportion of the time the specific event actually occurs. Here is important to notice that the *n* trials have to satisfy the three assumptions underlying the binomial distribution [Johnson05]:

- There are only two possible outcomes for each trial (arbitrarily called "success" and "failure").
- The probability of success is the same for each trial.
- The outcome from different trials are independent.

If these assumptions are met the theory following below is valid.

If the assumptions are true the mean and the standard deviation of the number of successes are respectively given by *np* and $\sqrt{np(1-p)}$. If both of the quantities are divided with *n* the mean and standard deviation of the sample proportion is found; respectively *p* and $\frac{\sqrt{p(1-p)}}{n}$. The first result shows that the sample proportion is an unbiased estimator of the binomial parameter *p*, exactly the true proportion which is the goal to estimate on the basis of a sample [Johnson05].

The magnitude of the error made when the point estimator *X/n* is used to estimate the real proportion *p* is given by:

$$\left| \frac{X}{n} - p \right|$$

Calling this magnitude *E* and using the assumption from above we get

$$E = z_{\alpha/2}\sqrt{\frac{p(1-p)}{n}} = z_{\alpha/2}\sqrt{\frac{\frac{x}{n}(1-\frac{x}{n})}{n}}$$

where $z_{\alpha/2}$ is the *z*-score for a given confidence level. Isolating *n* in the above formula gives an expression for the number of trails needed when a certain maximum error of the estimate is wanted, called the sample size determination:

$$n = p(1-p)\left[\frac{z_{\alpha/2}}{E}\right]^2$$

As seen from this formula it can not be used without some information about the size of *p*, exactly the quantity we want to estimate. So in this case no such information about *p* is available, and we have to make use of the

fact that *p(1-p)* is at most 1/4 (in the case *p*=1/2) which can be shown through simple calculus. The new determination of the sample size if *p* is unknown is:

$$n = \frac{1}{4}\left[\frac{z_{\alpha/2}}{E}\right]^2$$

Because *E<1* the above formula gives a rising number of trails *n* with a lower maximum error of the estimate of *p*. In the two tests of the success criteria is desirable to have as low error of the estimate of *p* as possible, but we have to take in mind the time aspect of having a high number of trial runs.

If we want to be at least 95% confident (z-table look-up: $z_{\alpha/2} = 1.96$) that the error of the estimate is at most 15 % (*E*=0.15) we need

$$n = \frac{1}{4}\left[\frac{z_{\alpha/2}}{E}\right]^2 = \frac{1}{4}\left[\frac{1.96}{0.15}\right]^2 = 42.68$$

or *n* = 43 rounded up to the nearest integer.

That means we have to run the test 43 times to estimate the proportion in each of the two use cases.

6.1.1.1 HYPOTHESES TEST ON ONE PROPORTION

Now everything is set to test if the two success criteria are met or not. The statistics test conducted is the hypotheses test on large samples (*n*>30) concerning *p*. The proportion on the sample test p_0 is tested with the null hypothesis $p = p_0$ against one of the alternative hypothesis p > $p > p_0$, $p < p_0$ or $p \neq p_0$ with the use of the statistic random variable:

$$Z = \frac{X - np_0}{\sqrt{np_0(1-p_0)}}$$

Below are the critical values for testing the null hypothesis $p = p_0$ in a large sample:

Alternative hypothesis	Reject null hypothesis if:
$p < p_0$	$z < -z_\alpha$
$p > p_0$	$z > z_\alpha$
$p \neq p_0$	$z < -z_{\alpha/2}$ or $z > -z_{\alpha/2}$

Test for the success criteria from Use Case 1:

Null hypothesis: p = 0.90
Alternative hypothesis: p > 0.90
Level of significance: alpha = 0.05
Criterion: Reject the null hypothesis if Z1 > 1.645 (t alpha table v = inf, alpha = 0.05), where

$$Z1 = \frac{X - np_0}{\sqrt{np_0(1-p_0)}}$$

So it can be concluded that the program detect more than 9 out of 10 cyclist in Use Case 1 if Z > 1.645.

Test for the success criteria from Use Case 4:

Null hypothesis: p = 0.10
Alternative hypothesis: p < 0.10
Level of significance: alpha = 0.05
Criterion: Reject the null hypothesis if Z4 < -1.645, where

$$Z4 = \frac{X - np_0}{\sqrt{np_0(1-p_0)}}$$

So it can be concluded that the program makes less than 1 out of 10 false positives in Use Case 4 if Z < -1.645.

6.1.1.2 DEFINITION OF A SUCCESSFUL DETECTION IN A CLIP

To test if the Success Criteria for Use Case 1 and 2 has been fulfilled, a criterion which determines the success of a single clip has to be defined. As stated in "5.3.5.1 Segmentation of Cyclist" the program issues a detection message for each frame. In order to convert the information about detection in one frame to a decision about detection in an entire clip it is decided to use the value “number of frames with a detection divided by the total number of frames”, which is calculated by the system for every frame. If this value never drops below a threshold in the clip the cyclist is successfully detected in Use Case 1 (True Positive), but with only one frame with a value below the threshold then the system fails to detect the cyclist (False Negative). In the same way if the value never exceeds the same threshold for Use Case 4 the system has successfully not detected a cyclist (True Negative), but with a single exception where the cyclist is detected and the clip is considered as a False Positive.

Now it is possible to divide the clip in two groups. One group where the system detects a cyclist, and one group where the system does not. The system is balanced by running it on the implementation footage from "5.2 Footage and Prerequisites" and tweaking based on the results, ending with a threshold value of 35% to be used for the Final Test.

With the statistics of the test defined it is now relevant to decide the more practical circumstances of the test.

6.1.2 TEST ENVIRONMENT

The input video for the test can be acquired in various ways. For example, it is possible to create a synthetic pre-rendered 3D environment to test on, like in "3.4 Test of Simple Dynamic and Static Systems". It is also possible to use real-world footage either by performing the tracking on pre-recorded real-world footage or by tracking live footage in real-time while the footage is being recorded.

Using a synthetic 3D environment would allow one to be able to control the testing conditions and e.g. retrieve ground-truth or gradually increase the difficulty of tracking all of which has both its advantages and merits. However, since the project is based off of a real-world problem it seems both relevant and interesting to test on footage recorded while driving and riding an actual car and bicycle.

As concluded in "5.3.5.2 Tweaking and Optimization" the system is not capable of doing all the necessary computations between each frame on the fastest laptop available for testing, before the next frame is ready from the video feed. This means that the frame rate must be reduced else the system will lag behind the live-feed or skip random frames. By using pre-recorded footage instead it will be possible to use the time needed to do the computations, because the next frame is queued until the system is ready to receive it. Therefore, the footage used for testing will be pre-recorded, and then afterwards the test will be run on the recorded footage.

The system has been designed to, but not implemented with, the ability to determine if a detected cyclist is in danger. This implies that the system computes the cyclist’s relative velocity to the car, and use this information to define a critical region where the cyclist was to be detected. Because the system does not have this feature,

the critical region must be defined beforehand on the test footage. In practical this implies that a vertical line, where the cyclist has to be detected from is applied to the footage. The system now has to be able to detect the cyclist when he reaches this line behind the car, until he exits the camera's field in the left side of the image. So the velocity of the car and cyclist must be decided beforehand.

6.1.2.1 VELOCITY OF CAR AND CYCLIST

In "3.1.5 Speed and Reaction Time" it was learned through a field test at what speeds a car takes a right turn, and from the report by the Investigation Team for Road Accidents the speed of the cyclists at time of impact was learned.

Looking at the speed of which the cars make the turn at it is apparent that there was a tendency leaning towards turning at 11-30 km/h (63 out of 71 cars). To cut down on testing scenarios one speed will be chosen for the car and the bicycle. For the car it would make sense to drive at 20 km/h since it is the midpoint between 11-30 km/h. Seeing as how the cyclist has to travel faster than the car to catch up with the car (as described in Use Case 1) the cyclist has to go faster than 20 km/h. To simplify matters and settle at a speed the cyclist will be travelling at 25 km/h in the test. Velocity will be monitored with a satellite navigation unit and a bicycle computer.

Since the testing will be performed with an actual car and bicycle (as opposed to e.g. using a custom made dolly rig on rails) it may prove challenging to accurately hold a specific speed especially for the car. Because of this the actual speed of the car and bicycle may end up varying slightly from the decided upon.

When deciding on the velocities of the car and cyclist, many other parameters in the Use Case 1 and 4 can be changed and tested on. Every time a parameter is changed, e.g. different lighting conditions, the required test sample to achieve same accuracy doubles. To reduce the amount of parameters and consequently test scenarios, the test will be performed without changing any parameter in the test scenario. The following is a list with the areas where parameters can be changed:

- Properties of the cyclist - type of bicycle, appearance of the cyclist, number of cyclists, etc.
- Properties of the background - type of landscape, non-cyclist IMOs (e.g. pedestrians), etc.
- Lighting conditions - time of day, weather conditions, etc.

In an attempt to mimic a real life situation it was decided not to attempt to control aspects beyond testing Use Case 1 and 4 with a single cyclist. This meant that no specific decisions were made for e.g. the type of bicycle, clothing, specific location, etc.

With the specifics of the Test Setup covered, testing commenced. What follows are the results of the test.

6.2 TEST RESULTS

In this chapter, the results from the test will be presented. The footage for the test was obtained on a small, undisturbed road with a similar background made up of hedges and similar houses. A Canon MiniDV MD215 camcorder was mounted in the foremost, right side window, angled at the mentioned specifications in "4.4.2 Amount of Cameras".

For Use Case 1, i.e. a cyclist catching up to and overtaking the car, 43 situations were recorded as specified by the determination of the sample size in "6.1.1 Statistics of the Test". One group member was driving the car while another was riding the bicycle. Since it was very difficult to maintain the same background in each frame (this would require identical velocities and starting points for each situation), the footage was shot with similar backgrounds, i.e. single-family houses and hedges. The speed of the car varied around 20 km/h and the speed

of the cyclist, when it was in the field of view, was approximately 25 km/h. For Use Case 4, i.e. with no cyclist in view, 43 situations were recorded, with the same backgrounds and speeds of the car as for Use Case 1. The results were retrieved after running the 86 video clips through the system. All videos can be found in [CDc] and data logs in [CDd]. An example of running the Test Footage through our sparse Lucas-Kanade system can be seen in Figure 45.

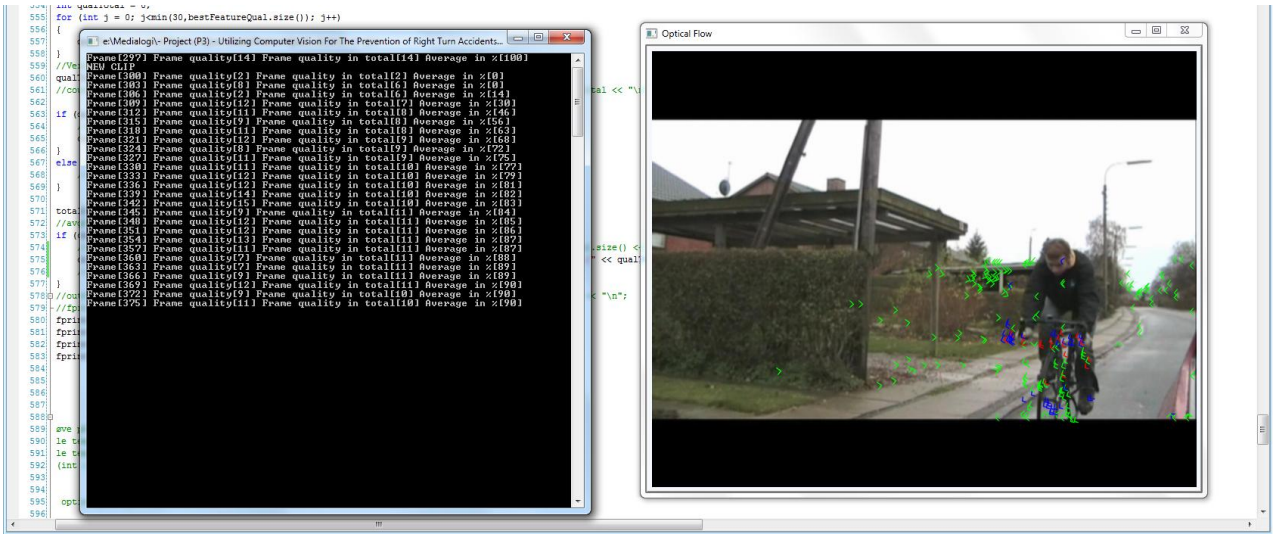


Figure 45: Screenshot from running the Test Footage through our sparse Lucas-Kanade system.

6.2.1 SUCCESS CRITERIA RELATED RESULTS

In Figure 46 the results from the Success Criteria test is shown. The figure tells the proportion of the clips where a cyclist was detected, from the definition in "6.1.1.2 Definition of a Successful Detection in a Clip".

Figure 46: Proportion of clips with a detected cyclist.

With the result from Figure 46 the hypotheses test on one proportion respectively from Use Case 1 and 4 can be carried out.

Recall that for Use Case 1 the hypothesis test was designed with:

Null hypothesis: $p = 0.90$
Alternative hypothesis: $p > 0.90$
Level of significance: $\alpha = 0.05$
Criterion: Reject the null hypothesis if $Z1 > 1.645$

$$X = 37, n = 43$$

$$Z1 = \frac{X - np_0}{\sqrt{np_0(1-p_0)}} = \frac{37 - 43 \cdot 0.90}{\sqrt{43 \cdot 0.90(1-0.90)}} = -0.86$$

Also for Use Case 4:

Null hypothesis: $p = 0.10$
Alternative hypothesis: $p < 0.10$
Level of significance: $\alpha = 0.05$
Criterion: Reject the null hypothesis if $Z2 < -1.645$, where

$$X = 9, n = 43$$

$$Z4 = \frac{X - np_0}{\sqrt{np_0(1-p_0)}} = \frac{9 - 43 \cdot 0.10}{\sqrt{43 \cdot 0.10(1-0.10)}} = 2.39$$

The conclusion on these Z-score values is found the "6.3 Discussion of Results".

6.2.2 RESULTS ON LEVEL AND QUALITY OF DETECTION

Use Case 1

Figure 47 shows the percentage of frames with cyclist detections in the end of each clip, e.g. in 22 clips (51% of total clips) more than 90% of the frames contained a cyclist detection. Since this figure is for Use Case 1, with a cyclist overtaken the car, the goal is to receive a high amount of detection. As it can be seen, 3 clips had 0-9% of frames with cyclist detections. The reason for this will be discussed in "6.3 Discussion of Results". The average percentage of frames with a cyclist detection, for all Use Case 1 frames, is 82% (2730 out of 3332 frames had a detection).

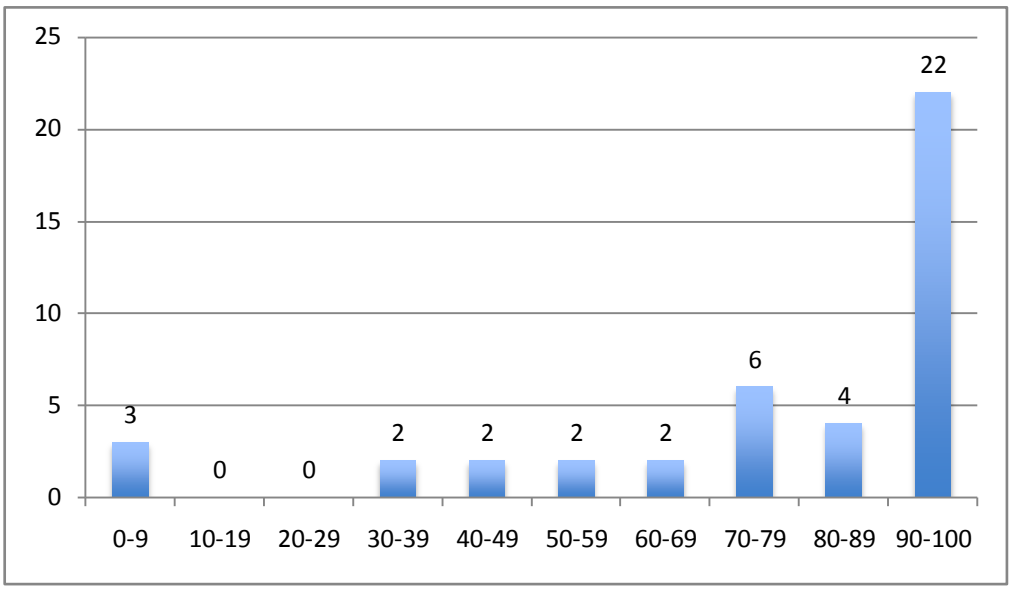


Figure 47: Percentage of frames with cyclist detection in the end of each Use Case 1 clip.

As in Figure 47, Figure 48 also shows the percentage of frames with a cyclist detection in each clip, e.g. in 29 clips (67% of total clips) 0-9% of the frames contained a cyclist detection. Since this Figure is for Use Case 4, with no cyclist in the field of view, the goal is to achieve the lowest amount of detections. Most clips are in the lower end of the scale, though 4 clips had a detection of a cyclist in more than 80% of the frames. The reason for this will be discussed in "6.3 Discussion of Results". The average percentage of frames with a cyclist detection, for all Use Case 4 frames, was 17.4% (622 out of 3569 frames had a detection).

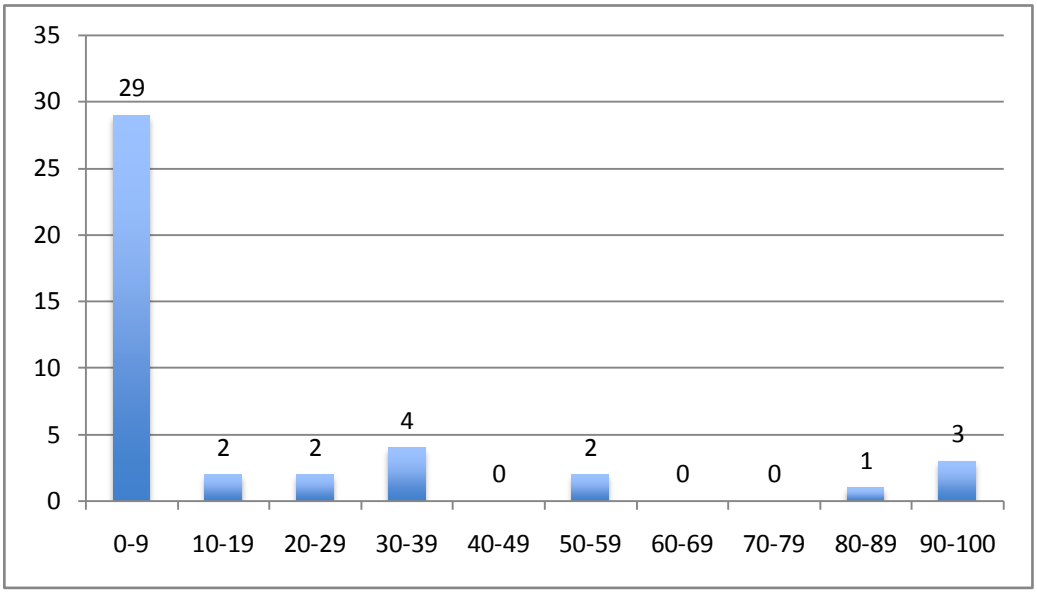


Figure 48: Percentage of frames with cyclist detection in the end of each Use Case 4 clip.

As mentioned in "5.3.5.1 Segmentation of Cyclist", a scoring system from 0-15 was used to categorize the quality of each cyclist detection. In the test the threshold of this was set to 6, meaning that all detections with a score of more than 6 is thought to be a cyclist.

If the decision between a True Positive or False Positive in Use Case 1 and True Negative or False Negatives in Use Case 4 should be taken for every frame and not the entire clip (as done in Figure 46) the distribution of the quality of the cyclist detection in each frame would be interesting to study. Figure 49 compares the presented

scores of the quality of the detection for Use Case 1 and 4 after the smoothing effect is applied (as mentioned in "5.3.5.1 Segmentation of Cyclist").

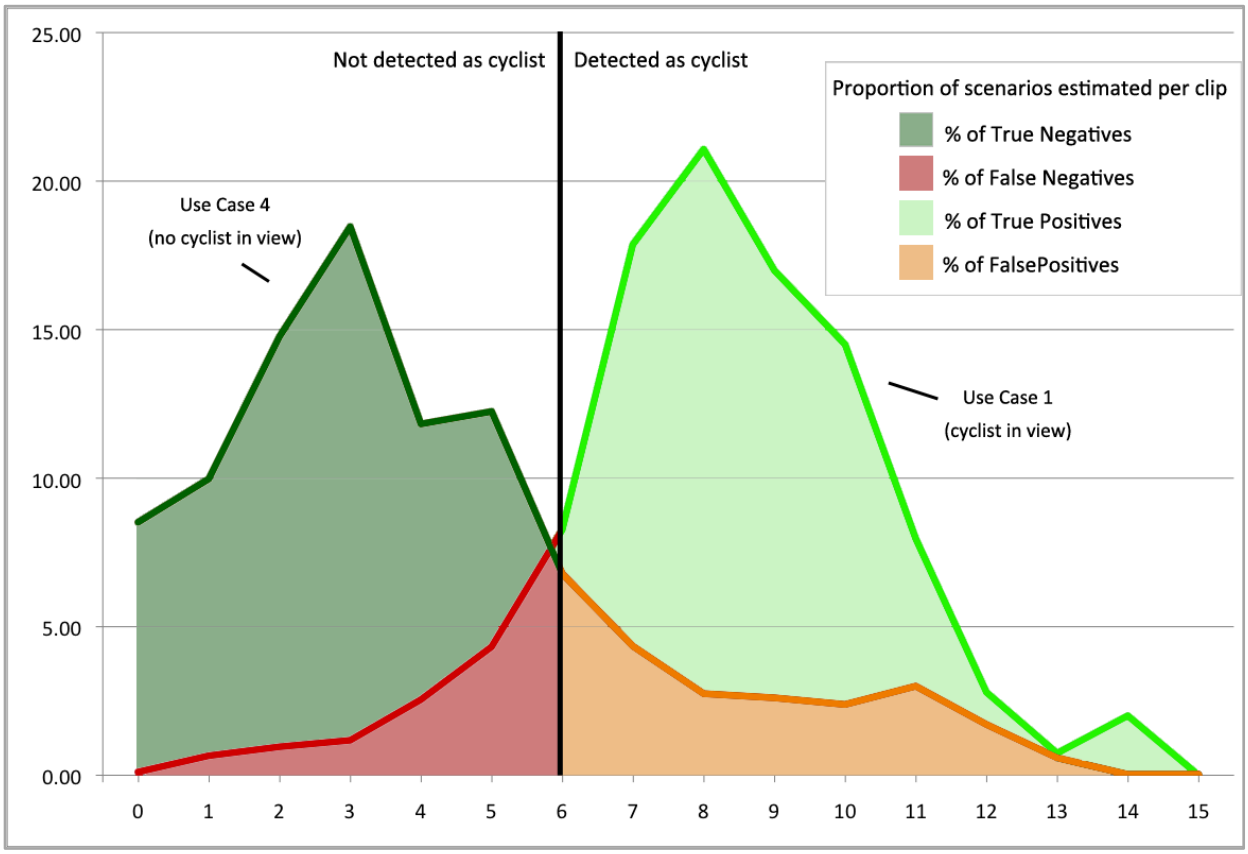


Figure 49: Proportion of scenarios estimated per clip for Use Case 1.
(Colors denote the areas under the curves.)

Figure 49 shows the scores for all 86 clips after smoothing, with the scores on the x-axis and the percentage of each score on the y-axis. Overall the dark green/orange curve is Use Case 4, i.e. no cyclist was in view, and the red/light green curve is Use Case 1, i.e. a cyclist was in view. The two curves denote the following results: The dark green is True Negatives, i.e. no cyclist is in view and no cyclist is detected. The orange curve shows the False Positives, i.e. no cyclist is in view, but a cyclist is detected. The light green curve is True Positives, i.e. a cyclist is in view and detected. The red curve denotes False Negatives, i.e. a cyclist is in view, but not detected. Use Case 1 peaks with 21% of its score points being 8. Use Case 4 peaks with 18.5% of its score points being 3.

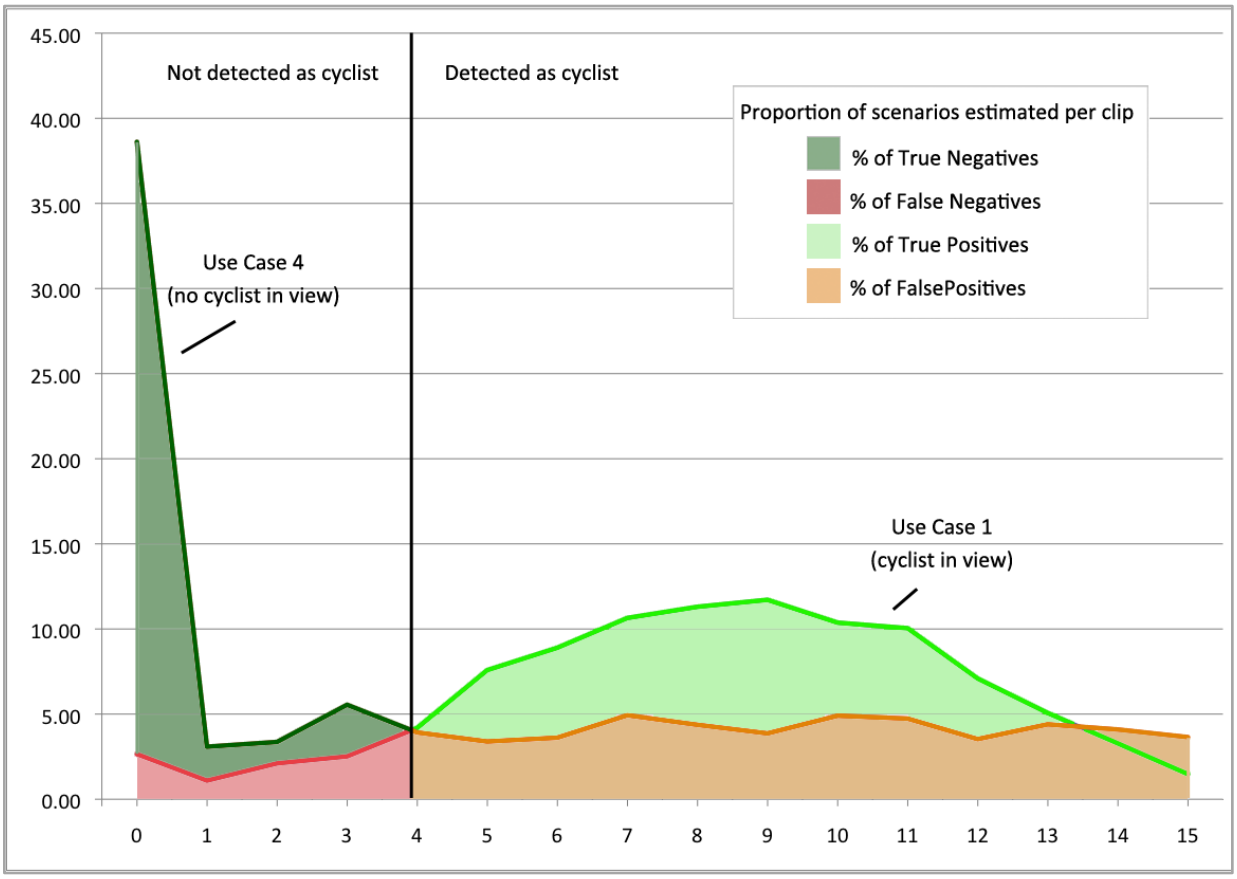


Figure 50: Proportion of scenarios estimated per clip for Use Case 4.
(Colors denote the areas under the curves.)

Figure 50 shows the total amount of scores for all 86 clips with no smoothing, with the scores on the x-axis and the percentage of each score on the y-axis. As with Figure 49 the dark green/orange curve is Use Case 4, i.e. no cyclist was in view, and the red/light green curve is Use Case 1, i.e. a cyclist was in view. The two curves denote the following results: The dark green is True Negatives, i.e. no cyclist is in view and no cyclist is detected. The orange curve shows the False Positives, i.e. no cyclist is in view, but a cyclist is detected. The light green curve is True Positives, i.e. a cyclist is in view and detected. The red curve denotes False Negatives, i.e. a cyclist is in view, but not detected. Use Case 1 peaks with 11.71% of its score points being 9. Use Case 4 peaks with 38.6% of its score points being 0.

Having now gone through the raw data it is relevant to look into whether the Success Criteria have been met, and why the results ended up being as they were.

6.3 Discussion of Results

To discuss whether the Success Criteria has been met it is relevant to go back and look at "6.1.1.1 Hypotheses Test on One Proportion", which defines how to determine the level of success.

Because Z1 = -0.86 is smaller than the critical value 1.645 is can be concluded that the system fails to meet the Success Criteria about detection of more than 9 out of 10 cyclists in Use Case 1. And as Z4 = 2.39 is larger than the critical value -1.645 is can be concluded that the system fails to meet the Success Criteria about detection in less than 1 out of 10 situations where there was no cyclist (Use Case 4). The test result is obtained with a maximum error on the estimate of the true proportion on 15% (as mentioned in "6.1.1 Statistics of the Test"), so it can be concluded that it is most likely that the test showed the true success rate of the system.

Now since the test showed that the Success Criteria was not fully met, it would be interesting to discuss what went wrong, but also to examine what was in fact a success. After all the system obtained a success rate of 37/43 (86.0%) correct detected cyclists for Use Case 1 and 9/43 (20.9%) wrongly detected cyclists in Use Case 4. The following is an elaboration of these facts.

For 22 clips in Use Case 1 more than 90% of the frames contained a cyclist detection, as seen and explained in Figure 47. The goal is obviously to receive a high amount of detection and it is clearly evident, that the detection rate in each clip is quite high. The average percentage frames with a cyclist detection, for all clips, was 82% (2730 out of 3332 frames had a detection). Even if the Success Criteria had been tested per frame as opposed to per clip it would still not have been fulfilled.

As mentioned, the percentage results for Use Case 1, as seen in Figure 47, showed that 3 clips only detected the cyclist in less than 9% of the frames. These clips were analyzed, because they might give an answer as to why the Success Criteria was not fulfilled. The reason for the low detection rate was that the cyclist was moving orthogonally towards the side of car, right when it was supposed to be detected, more specifically the cyclist's intersection points with the horizon was outside the range of the system's threshold. This was easily fixed and showed very good results for the True Positives. This change was also quickly tested on some sequences from Use Case 4, where it did not appear to give worse results than the Final Test. Another full test for this changed system would be needed to determine the complete ramifications of the system changes.

For 29 clips in Use Case 4 less than 10% of the frames contained a cyclist detection, as seen and explained in Figure 48. The goal is obviously to achieve the lowest amount of detections. As with Use Case 1 even if the Success Criteria had been tested per frame as opposed to per clip the criteria of detection in less than 1 out of 10 frames would not have been fulfilled.

The percentage results for Use Case 4, as seen in Figure 48, showed that 4 clips detected a cyclist in more than 80% of the frames. These individual clips were analyzed in order to state an explanation about as to why fulfilling the Success Criteria was not met. The reason for the high detection was due to very consistent background noise, meaning motion vectors from the background with more or less the same properties as the ones on a detected cyclist. It is doubtful that this special kind of noise would be possible to get rid off with simple tweaking and optimization of the system.

As seen in Figure 49 it is obvious that the system's threshold is very close to the optimal, which would be the intersection between the two graphs. However, the y-coordinate to this point is still too high, since the percentage of detections did not meet the Success Criteria. The lower the value of this point’s y-coordinate, the better the result. In Figure 49, Use Case 1 peaks with 21% of its score points being 8 and Use Case 4 peaks with 19% of its score points being 3. These peaks are rather close to the threshold of 6. The reason for this is that the system is smoothing the frames to get better results, which then also makes it less likely that the smallest and biggest scores will be chosen for frames. The goal of an optimization of the system would be to try to move these peaks away from the threshold and to lower the intersection's point on the y-axis, which then would decrease the light orange and light red areas, denoting False Positives and False Negatives.

The reason for including Figure 50 is that it interestingly enough shows that, for Use Case 4, a large percentage, 38.6%, of frames got a score of 0. This can be used to optimize the system, since this shows that many scores of 0 could indicate that no cyclist is in view. Comparing this system to Figure 49 also gives a good indication of how crucial, but also dangerous, the smoothing part of the system is, since it also moves a lot of the scores closer to the threshold. It is interesting to note that to balance this version of the system, the threshold should be placed at the score of 4 instead of 6, as it is with the smoothing part. This would yield a larger amount of True Positives, but also a significantly bigger amount of False Positives (detection in 87% of all frames for Use

Case 1, and detection in 46% of all frames for Use Case 4). This just shows that the system is much more well-balanced after the smoothing has been applied, since the percentages afterwards are 82% and 17% respectively.

Through testing it has been concluded that the Success Criteria set in "2.6 Success Criteria" were not met. Having said that the system was capable of reaching a successful detection rate in Use Case 1 of 37 out of 43 clips and 82% correctly detected cyclists as an average of all frames. In 9 out of 43 clips in Use Case 4 the system wrongfully detected a cyclist even though no cyclist was in the frame. When looking overall per frame the system wrongfully detected a cyclist in 17% of all the frames.

7. DISCUSSION

Even though "6.2 Test Results" showed we have not been able to fulfil the Success Criteria set in "2.6 Success Criteria", it must be said that we were ourselves a bit stunned during testing just how well the actual detection went. It came to the point where we questioned the validity of the data and whether the results would apply in a broader sense.

In all honesty during the span of the project as we delved into the more intricate features of optical flow and miscellaneous algorithms we have had our doubts about whether we would be able to put something together that would be functioning well enough to provide proper testing results.

As a result of this we have had to make a series of difficult choices to simplify the final test scenario. These simplifications have been rather extensive and in many cases we have had to simplify something, not because we wanted to, but because the alternative would be too time consuming.

We have had to lock the orthogonal distance to the bicycle at half a meter from the car in an attempt to mimic what we saw as a plausible and reoccurring "worst case" scenario. This in combination with a simplified distance measurement method means that whenever the bicycle strays from the straight line the system is likely to believe the bicycle is far advanced or farther behind the car than it actually is. This is a major flaw in the existing system, without precisely defined paths for the cyclists to follow the system will show considerable lower detection rate.

At its current state the program is most likely not able to function in conditions beyond flat terrain as we at one point saw it necessary to give up on coding an algorithm to dynamically find the horizon line and instead hard code a horizontal line. We also did not feel we had time to delve into the correlation between the car's velocity, the bicycle's velocity, and the two's velocity relative to each other. This means we are not sure how well the current program would scale if the velocity of any of the involved parties are changed.

During "3.5 Requirement Specification" it was found we have four different use cases. Primarily because we realized we would not be able to implement both the forward and the backwards pointing camera only Use Case 1 and 4 were addressed. Although a considerable part of the system was delimited it is still believed that the delimitations done allowed for an easy overview and implementation of the missing parts while pertaining the same general framework principles.

It can certainly be argued that all these cuts would have to have been made regardless of how well we would fare at implementing a solution. No matter how you look at it there are simply too many variables and what-ifs to cover them all in the span of a three and half months project – a project period significantly shorter than those of similar professional projects.

The lingering question is: Did we create an actual working program for any right turn situation or did we merely create a program that is tweaked to detect a bicycle on the specific eighty-some clips we recorded for the test?

The correct answer must lie somewhere in between. The program was quite successful at detecting cyclists, but left room for improvements. We created a program which worked satisfactorily under the strict framework of the report, but no proper segmentation has been designed to cater to the ever changing conditions in the Danish traffic.

It was never a goal to create a product meant to work under all conceivable conditions. To us the purpose of the product went beyond the practical application of the product itself. 1 We did not choose what appeared to be the easier path of going with a static solution with, a statically mounted camera. Choosing a static solution

would have meant that optical flow would not be needed, and with a primarily consistent background it would have been considerably easier to detect whenever movement entered the frame. Altogether this was also the exact reason why we chose the dynamic solution: We liked the challenge it posed. We liked how we were not entirely sure how we would effectively be able to fulfil the Success Criteria.

We had gotten the impression from our research of optical flow that the principles behind dense algorithms might solve the possible scenario of what if the Shi-Tomasi algorithm did not find any features rendered important enough to track. Since we are working with a real world problem, where if applied the implications of the choices made in the project would have a considerable impact in a real world situation, it made sense to go with whatever would be most consistent and accurate

Having said that, it is almost certain that we would have ended up with a sparse Lucas-Kanade with a more well-tweaked segmentation had we focused all our efforts on that from the beginning. However, by going with sparse Lucas-Kanade from the beginning would have meant going with an optical flow we knew to work. Since we wanted to learn, we attempted to go further. The same can be said when the choice had to be made between a dynamic or static solution. Both solutions have their merits, but again we felt there were more unanswered questions by throwing ourselves towards optical flow and a dynamic solution. In no way would a static solution have been a walk in the park, but to us the journey there seemed less challenging.

8. CONCLUSION

Truth be told the problem area we chose is not unique. We have not ventured into uncharted waters, and many people far more skilled than us have attempted and succeeded in what we have merely scratched the surface of. It is very likely that several scientists and engineers are working on or have already solved the issue of preventing collisions with vulnerable road users - and not necessarily limited to right turn situations.

From the outset we knew we were never going to pioneer a new set of algorithms. Nor were we likely to present a new combination of existing methods in solving our problem. At best we would be able to combine the work of others and tailor it to the specific needs outlined by the project. The challenges would have to be found in the choices made throughout the project.

Now, at the end of the project a conclusion has to be drawn. Looking at what went wrong and what went right should give an impression of how well you succeeded in what you set out to do. The final problem statement for this project has been:

How can one prevent collisions between cars and cyclists in right turn situations by the use of computer vision?

The theme for this semester has been “Human senses – Digital perception”. Three and a half months ago both computer vision, programming in C++, and using the tools offered by the OpenCV libraries were new territory to us. Looking back it can easily be argued that computer vision has been covered as we have been able to get optical flow up and running successfully – we made the computer “able to see”.

Within the framework of the project, what we failed at in the end was properly defining and setting up the parameters for the segmentation. We were not able to consistently detect the cyclist within the set success criteria. To put this failure into perspective it would be relevant to note that defining how to properly segment the cyclist from the background is an area that touches upon artificial intelligence; the principles of which are taught on later semesters.

Had the focus of the final problem statement evolved around “detecting a cyclist” we would have been closer to successfully answering the final problem statement, but arguably also further away from the initiating motivation of preventing accidents.

To actually prevent any collision the driver needs to be warned and, if it comes to it, the car has to be stopped. The cynical reader would be right in pointing out that we have not prevented anything in this project. At best we have detected the cyclist, and in a sense, detecting the cyclist is only half the journey towards preventing collisions. It is, however, in our opinion the more challenging and interesting stretch of the journey.

9. BIBLIOGRAPHY

All references will contain author, title and if available publisher, year published and ISBN, and will appear in this order. They will be listed in alphabetical order. Website references are explicitly stated as such and can be found in downloaded form on the attached CD in the References folder under the corresponding reference.

[AMSalpha] American Mathematical Society, *"BibTeX bibliography style amsalpha"*, 1995.

[Anderson99] Scott Anderson, *"Image Stabilization: EIS/OIS"*, 1999. Website.

[Biering-Sørensen88] Stephen Biering-Sørensen, Finn Overgaard Hansen, et al., *"Håndbog i struktureret programudvikling"*, 1988. ISBN: 978-87-571-1046-3.

[Bradski08] Gary Bradski and Adrian Kaehler, *"Learning OpenCV"*, 2008. ISBN: 978-0-596-51613-0.

[CDa] *"Interview with HVU chairman, Sven Krarup Nielsen"*. On CD.

[CDb] *"Measurement of car specifications"*. On CD.

[CDc] Videos from Test. On CD.

[CDd] Data logs from Test. On CD.

[Commission06] European Commission, "On the Intelligent Car Initiative: Raising awareness of ICT for Smarter, Safer and Cleaner vehicles", COM(2006) 59 final.

[Damien09] Damien, *"Optical Flow - Gunnar Farnebäck's Algorithm"*, 2009. Website.

[Flair07] Lui Flair, *"Horn & Schunck Example"*, 2007. Website.

[Gandi08] Tarak Gandhi and Mohan Manubhai Trivedi, *"Computer Vision and Machine Learning for Enhancing Pedestrian Safety"*, 2008, ISBN: 978-3-540-79256-7

[Gavrila06] D. M. Gavrila and S. Munder, *"Multi-cue Pedestrian Detection and Tracking from a Moving Vehicle"*, 2006.

[Gerónimo07] David Gerónimo, Antonio López, et al., *"Computer Vision Approaches to Pedestrian Detection: Visible Spectrum Survey"*, 2007 . ISBN: 978-3-540-72846-7.

[Hansen05] Finn Overgaard Hansen, *"SPU-UML note Systematisk Programudvikling med UML"*, Ingeniørhøjskolen Aarhus, 2003.

[Havarikommissionen06] Havarikommissionen, *"Ulykker mellem højresvingende lastbiler og ligeudkørende cyklister"*, 2006. ISBN: 87-91458-10-2.

[Havarikommissionen08] Havarikommissionen, *"Krydsulykker mellem cykler og biler"*, 2008. ISBN: 978-87-91458-04-0.

[Heikkila96] J Heikkila and O Silven, *" Proceedings of the 13th International Conference on Pattern Recognition, volume 1"*, 1996. ICPR.1996.546012.

[Horn81] Berthold K.P. Horn and Brian G. Schunck, *"Determining Optical Flow"*, Artificial Intelligence: Volume 17, 1981.

http://ec.europa.eu/information_society/activities/intelligentcar/technologies/tech_04/index_en.htm

[icar06] Icar, *"Night Vision Enhancement System"*, 2006.

[IEEE98] IEEE, *"Recommended Practice for Software Requirements Specification"*. The Institute of Electrical and Electronics Engineers, 1998. ISBN: 0-7381-0332-2.

[Jung09] Ho Gi Jung and Jaihie Kim, "Constructing a pedestrian recognition system with a public open database, without the necessity of re-training: an experimental study", 2009.

[Kanade81] Bruce D. Lucas and Takeo Kanade, *"An iterative image registration technique with an application to stereo vision"*, Proceedings of Imaging understanding workshop, 1981.

[Komm09] Færdselssikkerhedskommissionen *"Baggrund for nedsættelse af Færdselssikkerhedskommissionen"*, 2009.

[Kohoutek08] Peter Kohoutek, *"Der neue Audi A4: Entwicklung und Technic"*, 2007. ISBN: 978-3-8348-0399-3

[Lucas85] Bruce David Lucas, *"Generalized image matching by the method of differences"*, 1985.

[Murakami04] Ikuya Murakami, *"The aperture problem in egocentric motion"*, 2004.

[Modrow07] Daniel Modrow and Claudio Laloni , *"3D Face Scanning Systems Based On Invisible Infrared Coded Light"*, 2007. ISBN: 978-3-540-76857-9.

[Marsden00] Greg Marsden, Mike McDonald1, et al., *"Towards an understanding of adaptive cruise control", 2000.*

[Moeslund08] Thomas B. Moeslund, *"Image and Video Processing"*, 2008. ISBN: 978-87-992732-0-1

[Marzat09] Julien Marzat, Yann Dumortier, et al. *"Real-Time Dense and Accurate Parallel Optical Flow using CUDA",* INRIA Rocquencourt, 2009. ISBN 978-80-86943-93-0.

[Niewoehner05] Walter Niewoehner, F. Alexander Berg,, *"Endangerment of pedestrians and bicyclists at intersections by right turning trucks"*, The 19th International Technical Conference on the Enhanced Safety of Vehicles (ESV), Paper Number 05-0344, 2005.

[Nielsen09] Ref til mail fra Sven Krarup Nielsen

[OpenCV09] Willow Garage, "OpenCV 2.0 C++ Reference: Motion Analysis and Object Tracking", 2009.

[Qiu08] Shao-Qiu Xiao, Ming-Tuo Zhou, et al., *"Millimeter Wave Technology in Wireless PAN, LAN, and MAN"*, 2008. ISBN: 978-0-8493-8227-7

[Reitzel09] Hans Reitzel , *"Interview: kunsten at lytte og spørge - En Introduktion Til Det Kvalitative Forskningsinterview"*, 1st edition, 2006. ISBN-13 978-87-412-2816-7.

[RSTA08] Færdselsstyrelsen, *"Sensorer som kan opdage cyklister og fodgængere"*, 2006.

[SAVEU05] P. Marchal, D.M. Gavrilla, *"SAVE-U: An innovative platform for vulnerable user proection"*, 2005.

[Sayed09] R. Sayed and A. Eskandarian, *"Unobtrusive drowsiness detection by neural network learning of driver steering"*. Proceedings of the Institution of Mechanical Engineers: Volume 215, 2001. ISBN: 0954-4070.

[Schneider05] Martin Schneider, *"Automotive Radar – Status and Trends"*, 2005.

[Shi94] Jianbo Shi and Carlo Tomasi, *"Good Features to Track"*, In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR94), 1994.

[Sikkertrafik07] Sikkertrafik, *"Hvor er hvor tit er uheldet ude?"*, 2007. Website.

[Statistik08a] Danmarks Statistik, *"Tilskadekomne og dræbte i færdselsuheld"*, 2008. Website.

[Statistik08b] Danmarks Statistik, *"Færdselsuheld 2007"*, 2008. ISBN: 978-87-501-1708-7.

[Stavens07] David Stavens, *"Lecture Notes on Optical Flow and OpenCV"*, Stanford University, 2007.

[Stonerain08] Stonerain, *"Block Matching Example"*, 2008. Website.

[Turner-Fairbank97] Turner-Fairbank Highway Research Center, *"Bicycle Crash Types: A 1990's Informational Guide"*, Publication no. FHWA-RD-96-104, 1997.

[USC] USC Viterbi School of Engineering, *"USC-SIPI Image Database"*.

[Vaudrey08] Tobi Vaudrey, Clemens Rabe, et al, *"Differences Between Stereo and Motion Behaviour on Synthetic and Real-World Stereo Sequences"*, in Proc.23rd Int. Conf. Image and Vision Computing New Zealand, Nov 2008.

[Welch06] Greg Welch, Gary Bishop, "*An Introduction to the Kalman filter*", TR95-041, 2006.

[Wolfe09] Wolfe, Kluender, Levi et al., "*Sensation and perception, second edition"*, Sinauer, 2009.

[Zhang07] Fan Zhang, Jens Sørensen, et al., *"Trucks With Eyes"*, MED3, AAUK, 2007.