Machine to Machine Perception for Safer Traffic.

Gijs Slijpen

s1978004

January 2013

Master Project

Artificial Intelligence

University of Groningen, The Netherlands

Internal supervisors:

Dr. Marco Wiering (Artificial Intelligence, University of Groningen)

External supervisor:

Dr. Wang Chieh-Chih (Computer Science, National Taiwan University)

university of groningen / faculty of mathematics and natural sciences / artificial intelligence

Contents

Abstract

Participation in traffic requires drivers to perform multitasking. A driver needs to pay attention to different events in traffic occurring at the same time. Other events, like a phone call, or people having a conversation in the back seat, can draw the driver's attention away from what is important. This thesis proposes a framework that implements a safety warning system that uses machine to machine communication to communicate with other traffic users. The safety warning system seeks to aid traffic users with the task of detecting possible dangers during participation in traffic. The system consists of four different parts: perception of the environment, the communication protocol, the localization and tracking system, and a collision risk assessment algorithm. Perception of the environment is performed by using a laser-rangefinder. The localization and tracking system keeps track of a traffic user's own location and the locations of other moving objects with respect to the traffic user itself. A single-hop broadcasting communication protocol between traffic users enhances the range of the localization and tracking system by sharing the detected moving objects with other traffic users. By using Multiple Hypotheses Tracking (MHT), measurements of the same moving object obtained by different traffic users are associated with each other. The localization and tracking system is tested by using three robots that are equipped with several sensors including a LIght Detection And Ranging (LIDAR) sensor and an Inertial Measurement Unit (IMU). The collision risk assessment algorithm consists of a Support Vector Machine (SVM) that is trained and tested using simulator data. The simulator aims to create data as realistic as possible by using a physics engine. It is shown that a collision can be detected 2 seconds before the collision occurs.

Chapter 1

1 Introduction

With an increasing amount of traffic, traffic accidents are becoming more common. An estimated 1.2 million people die each year as the result of traffic accidents. It is thereby estimated that another 50 million people get injured every year because of traffic accidents [38]. In 2011, 661 people died during a traffic accident in the Netherlands alone, in countries with less well formulated traffic rules these numbers are significantly higher.

Participation in traffic requires multitasking as many events take place at the same time. When driving a car, a traffic light might turn orange while pedestrians cross a red sign and a biker that was first visible in the mirror has suddenly disappeared. A car driver is often not only paying attention to the road, but also to events occurring within the vehicle. Passengers might be talking on the backseat, the radio might draw the driver's interest or the driver might be involved in a (hands-free) phone call. Pedestrians in their turn need to pay attention when crossing the street, but might at the same time be caught up in a vivid conversation or be checking their phone. It is easy to see that little is needed to distract someone from the main tasks of paying attention to traffic.

A safety warning system seeks to aid traffic users with the task of detecting possible dangers during participation in traffic. Such a system therefore needs to give feedback to the user when he or she finds himself in a dangerous situation. Eventually, the technology behind a safety warning system can become part of a greater system that autonomously drives a car. The system can indicate when a car should use more throttle, when it should break, and when it should steer in case of a dangerous situation. The purpose of autonomous cars is not only to create safer means of transportation, they also take away the need to have a driver, therefore giving people more time to relax or work while traveling. A safety warning system needs to be able to perceive the environment to be able to make judgements about the current traffic situation, it needs to be able to predict situations that might follow from the current situation and at last it needs to determine if the current situation is dangerous.

In this thesis a framework is proposed for an Intelligent Transportation System (ITS) that implements such a safety warning system. This system consists of four parts: perception of the environment, communication, localization and tracking, and collision risk assessment. For perception of the environment various sensors are utilized. Of these sensors the Light Ranging and Detection (LIDAR) sensor and an Inertial Measurement Unit (IMU) are used for localization and tracking. Three different robots are configured to carry these sensors and to perform outdoor measurements. Sensors are sensitive to noise and their measurements therefore lead to imprecise localization and tracking of moving objects. By communicating with each other, traffic users share localization and tracking data of themselves and moving objects they detected. Moving objects that are visible for only a few traffic users can in this way become visible for all traffic users that are in the same communication group and sensor noise can partly be filtered out.

Two Machine to Machine (M2M) communication protocols are proposed in this thesis. The first protocol only enables direct neighbors to communicate with each other by using a one-hop broadcasting protocol. The second protocol allows for messages to be broadcasted further than only to the direct neighbors of a sender, therefore increasing the range at which traffic users can communicate. The received data is merged together with the traffic user's own data by using Multiple Hypotheses Tracking (MHT) [40]. After the data is merged, a map is obtained that contains the locations of each moving object with respect to the traffic user.

The localization and tracking part of the framework is responsible for the localization and tracking of the traffic user itself and the moving objects detected by the traffic user. The LIDAR-sensor generates multiple scans per second. These scans consist of range measurements defined in a polar coordinate system. By aligning and merging scans obtained at consecutive time steps, a greater map can be created. In order to align the scans, it is required that they have a certain similarity. The amount with which one scan needs to be rotated and translated to align with a scan from a previous time step indicates the rotation and translation of a traffic user during a certain period of time. The point cloud of one scan can be segmented into smaller point clouds called segments. Similar segments in two different scans that only differ in location can indicate moving objects. A Kalman filter [24] using two different prediction models is deployed to track traffic users and moving objects.

The collision risk assessment algorithm predicts if a collision might occur at a certain point in the future. This algorithm consists of a Support Vector Machine (SVM) [1] and uses several different features as input. The collision risk assessment algorithm is trained and then tested in order to evaluate its performance. While it is possible to use the localization and tracking system to create a data set for training and testing, a simulator was used in this project. There are several reasons for using a simulator over the localization and tracking system: a simulator is safer than letting real traffic users crash into each other, the environment in a simulator is easier to control, and large training and testing sets can be generated in reasonable time.

The simulator presented in this thesis uses two simulated cars to generate data. Each session, two cars follow a predefined path while adding noise to the input (i.e. steering and throttle), each path is thus always unique. A session starts with the two cars at their respective starting position and ends when both cars finish their predefined path or when they crash into each other. The paths both cars drive are stored and a label is added. When the cars collide with each other the paths are classified as dangerous and otherwise as safe. By using the path that was driven by both cars, features such as the relative distances, speeds and rotations between the two cars at each time step can be extracted. There are six features in total, the label and the amount of features multiplied by the amount of time steps in a session represents one data point. Multiple sessions are executed to generate a data set for training and testing of the algorithm.

Even though during this project the training and testing data is generated using a simulator, the environment perception, communication, and localization and tracking parts are of great importance. When the different parts are eventually merged into one system, the localization and tracking part will provide the input for the collision risk assessment algorithm that is now generated by the simulator.

One of the objectives of the safety warning system is that it is applicable for many kinds of transportation methods used in traffic. In this project the localization and tracking part of the safety warning system are evaluated using robots, while the collision risk assessment part is evaluated using simulated cars. In this thesis all these different kinds of transportation methods and the robots are referred to as traffic users. It is assumed that all traffic users have the safety warning system at their disposal. This assumption does not necessarily hold for all moving objects, however the term moving objects can include both traffic users and other objects that are moving. It can take time for a traffic user to determine if an object that is moving is another traffic user or a moving object, during this time the object is considered to be a moving object.

By implementing the above proposed framework two research questions are attempted to be answered:

1. Can a Support Vector Machine predict upcoming dangerous situations?

2. If the answer to question 1 is positive: What is the predictive capacity of the classifier when using the two second rule?

The 2-second rule is a rule of thumb used by many governments to instruct drivers on how to keep distance between cars [48]. It is assumed that 2 seconds are enough to either avoid an accident or to reduce the impact of an accident significantly.

This thesis starts with an explanation of related work in chapter 2. Chapter 3 elaborates on the different robots used for testing, the sensors that are used and the communication protocol that is used to share the sensory information. Chapter 4 explains how ego estimation is performed and how moving objects are tracked. Chapter 4 thereby elaborates on how moving objects detected by different traffic users are merged into one map using multiple hypothesis tracking. Chapter 5 explains the implemented simulator and how this simulator is used to obtain training and testing data to train and test the classifier with. Chapter 6 describes the implemented classifier, it explains the basic theory of the classifier, how data is preprocessed for classification and elaborates on the results obtained. In the last chapter a discussion is given and possible future work is discussed.

Chapter 2

2 Background and Related Work

This chapter gives an overview of related work on different parts of the framework described in this thesis. First, previous work on machine to machine communication is elaborated on in section 2.1. Previous work on localization and tracking is described in section 2.2, and in section 2.3 some background and previous work on safety warning systems is discussed.

2.1 Communication

To make relative localization and tracking more effective Machine to Machine (M2M) communication is used. In the context of this thesis M2M communication is the communication between different cars. This section explains related work in M2M communication between cars and possible infrastructure nodes. A Vehicular Ad-Hoc Network (VANET) is a class of networks that includes any ad-hoc network between two or more moving vehicles, VANETs are mainly used in intelligent transportation systems [27].

Karthikumar and Krishnaveni describe different common entities for VANETs in [26]. The first entity they describe is a VANET used for infrastructure purposes. Such a VANET can be used to communicate between a traffic control center and cars. Information about road conditions, traffic jams and accident locations are conveyed to a traffic control center. This information can then be used to notify other drivers using dot-matrix displays or the same VANET.

In Vehicle to Vehicle (V2V) group communication only vehicles that are of the same group can communicate with each other. These groups can be defined statically (e.g. grouped by car brand) or dynamically (e.g. when vehicles are in the same area during a certain time interval). Group communication can for example be used for entertainment purposes. In vehicle to vehicle beaconing, periodical messages are sent out to nearby vehicles and convey information such as speed, heading and braking. These messages are sent out to increase neighborhood awareness.

The last entity Karthikumar and Krishnaveni describe is the infrastructure to vehicle (I2V) and the vehicle to infrastructure (V2I) warning entity. In case of potential danger, messages are broadcasted by either an infrastructure node installed on the side of the road or by a vehicle. A roadside unit can for example send out a message to a car when it is approaching an intersection while a possible collision might occur.

Maihöfer describes different geocasting routing protocols including the Location Based Multicast (LBM) geocast routing protocol in [29]. The LBM protocol avoids flooding the whole network by using a forwarding zone. When a package travels outside the forwarding zone the package will be discarded. Such a forwarding zone can for example be defined using GPS coordinates. The protocol fails however when the forwarding zone is partitioned.

Bachir and Benslimane describe a multicast protocol for vehicular ad hoc networks in [4]. An area in which an accident has happened or a vehicle broke down is defined as a risk area. Vehicles are inside a risk area if they are driving towards the accident. Whether a vehicle belongs to a multicast group (the forwarding area) depends on the vehicle's location, speed and driving direction. This reduces the amount of traffic on the network since automobiles for whom the message is not applicable, but who are in range of the network, will not receive an alarm message.

Ye, et. al. describe a collision avoidance system in [53] that uses V2V communication for avoiding rear-end collisions on highways. Their communication protocol assumes that every vehicle on the highway is equipped with a positioning device (e.g. GPS) and IEEE 8.02 radio. Emergency Warning Messages (EWM) are sent out in case of an emergency. Such a message contains the sender's position, a lane ID, an event ID, event location, an event time stamp, and the message's lifetime. These messages are sent out using multi hop broadcast. A message is accepted only when it was received from the vehicle in front with the same lane ID, the event ID is new and the message has not exceeded its lifetime.

2.2 Localization & Tracking

Much research has been done in the field of localization and tracking, a few papers are discussed below. Premebida et. al. [39] describe a system for pedestrian and vehicle detection and tracking using a light ranging and detection (LIDAR) system and a camera. By using the LIDAR, detection and tracking is performed while object classification is performed by both the LIDAR and the camera system. After segmentation and data association a linear Kalman filter is employed for tracking of cars and pedestrians. Data points obtained from LIDAR scans are classified using a Gaussian mixture model. Objects detected by the camera system are in turn classified using an AdaBoost classifier. This system is able of classifying pedestrians and cars in various positions up to a distance of 20 meters.

Wang et. al. [50] describe a mathematical framework which aims to integrate Simultaneous Localization And Mapping (SLAM) with tracking of objects. The paper describes two implementations: SLAM with generalized objects and SLAM with Detection And Tracking of Moving Objects (DATMO). SLAM with generalized objects is computationally very expensive as it keeps track of both static and moving objects by maintaining posteriors for both. Due to the computational demands of this technique it is generally infeasible to use it in real-time. The paper therefore also describes an implementation in which only moving objects are tracked. By only detecting and tracking moving objects, a lower dimensional space can be used posing less computational requirements on the system. The paper proposes a framework in which perception modeling, data association and moving object detection are described. During perception modeling the input obtained from sensors is preprocessed by segmenting the laser points. Then, during data association a multiple hypotheses method is used in which multiple hypotheses of data association are generated and the most likely hypothesis is chosen. In moving object detection, laser point segments that are associated with the same object and of which at least half of all points in the segment have moved are classified as a moving object. A track is initiated for each newly detected moving object.

Competitions such as the DARPA urban challenge have been set up to stimulate research in the field of localization and tracking. An example of a project that was set up to compete in the DARPA urban challenge is Junior, an autonomous vehicle. Junior is described by Montemerlo et. al. in [34]. Equipped with a variety of sensors such as laser scanners and radar the vehicle needs to find its way in an urban environment. For this, path planning is required, thereby, a collision avoidance system is implemented to avoid collisions with static and dynamic obstacles such as curbs and cars. When an obstacle is detected the car plans a path around it. The goal is to arrive at the finish line before the cars of other teams do while obeying all traffic rules. The paper describes in detail how environment perception of the car, the software, mapping, tracking and localization is implemented.

2.3 Safety Warning Systems

Many of the documented safety warning systems are static systems that are positioned alongside the road. Such systems are described by Gangisetty in [16] and by Kamijo et. al. in [25]. Both describe static systems that are positioned at crossroads. Dagan et. al. [11], Miller & Huang [31], Morioka et. al. [35] and Seiler et. al. [45] propose the use of thresholding to determine if a possible collision might appear.

Salim et. al. [42] propose a collision detection system based on a C4.5 decision tree that uses information from roadside sensors and in-vehicle sensors. Training and testing data to train and test the decision tree with is obtained using a 2D traffic simulator. Other proposed implementations use more complex algorithms. Edgar & Harris [14] describe a neural network called the Cerebellar Model Articulation Controller (CMAC) and the Conventional Linear Model (CLM) to classify dangerous situations.

Kamijo et. al. describe a system in [25] that uses a spatial-temporal Markov random field (MRF) which is trained using a Baum-Welch algorithm. Training data consists of camera images containing observation sequences of real accidents that occurred at different intersections.

Zou et. al. [54] use camera images as data, Principal Component Analysis (PCA) for feature extraction, and Hidden Markov Models in combination with a Support Vector Machine (HMM/SVM) classifier to detect traffic incidents. First, images are resized then features are extracted while at last these features are classified using the HMM/SVM classifier. To train the classifier, 100 images for each class are sampled. These classes are defined as traffic movements, namely: West-East, East-West, North-South, South-North, and accidents. Then another 100 images are sampled for each class to test the classifier with.

Huang & Lin propose an algorithm called: Early Collision Warning Algorithm (ECWA) in [21]. This system is omni-directional and therefore provides a 360° range of protection and early warnings for drivers. In the early collision warning algorithm, Relationship Information (RI) packages are broadcasted to neighbors. Information within an RI message contains measurements of GPS and accelerometers. The ECWA deploys a Closest Point of Approach denoting the point at which two vehicles will pass by each other while having the smallest distance to each other. The distance from a vehicle A to this point is denoted as DCPA while the variable TCPA denotes the time it takes to reach the Closest Point of Approach for both vehicles. When DCPA is equal to zero a collision might occur while TCPA then denotes the time to this collision. By using thresholding it is decided if a warning message needs to be sent out.

Aköz & Karsligil [3] describe an implementation of a traffic abnormality detection. Using a HMM and a mixture of Gaussians (MoG) trajectories of traffic are clustered. These trajectories represent the normal traffic situation and are also called models. After training, for every vehicle observed the log-likelihood is determined of that vehicle following a certain model. If no model can be matched to the trajectory of the observed vehicle, the behavior of the vehicle is classified as being abnormal. Such an abnormal situation can for example represent an accident.

Misener & Sengupta introduce the term Cooperative Collision Warning (CCW) in [32]. They show that wireless communication between cars does enhance safety given that all cars have a wireless communication module installed. Three different systems are proposed in their paper: a forward collision warning system, an intersection warning system, and a blind-spot and lane change warning system. Their system uses GPS signals and a fusion of onboard sensors to measure wheel speed and yaw rate. Also a vehicle dynamics model is used to predict vehicle movements. By using a communication system the values of the different sensors are broadcasted to neighboring vehicles in order to obtain a 360° degrees awareness. Possible threats are displayed to the driver using an in-car display.

2.4 Discussion

The papers discussed in the previous section all implement a safety warning system. Where some of them use simple methods such as thresholding, others use more complex algorithms such as hidden Markov models. However, most systems are only designed to work on a predefined intersection or highway. The papers that describe a communication enhanced safety warning system only share GPS data, and only detect frontal and rear collisions. The papers that describe a system based on thresholding often do not need training data and testing is done using real cars. The papers that describe a more complex system either use image sequences of one intersection or use a 2D simulator to generate data. This thesis describes a system in which LIDAR data can be shared using communication and in which a 3D car simulator with physics is used to generate training and testing data. Because of safety issues and because the scale of the project is too small at this moment, the LIDAR data described in this thesis is not yet used for training and testing of the collision risk assessment algorithm. The safety warning system developed in this project can however be adopted to incorporate LIDAR data for training and testing purposes.

Chapter 3

3 Robots, Sensors, and Communication

This chapter introduces the hardware used during this project. First the robots that are used to test the localization and tracking system are explained in section 3.1. Then the sensors that are used to obtain measurements are explained in section 3.2, at last the communication protocol with which data is shared amongst traffic users is explained in section 3.3. This last section also discusses a second, not implemented, communication protocol.

3.1 Robots

To test the localization and tracking described in chapter 4, robots are used instead of real transportation methods for several reasons:

1. Robots are cheaper than using real means of transportation.
2. Robots can carry sensors while this can be difficult for some means of transportation.
3. Robots can easily be modified.
4. Robots are a reasonable simplification.

Each element in the list will shortly be explained below.

1. Real means of transportation include pedestrians, bikes but also cars, busses and trucks. However, using a car to transport the equipment is a more expensive solution than using simpler robots. In an early stage of the project it is acceptable to use robots for development and testing to get initial results of the performance of the system.

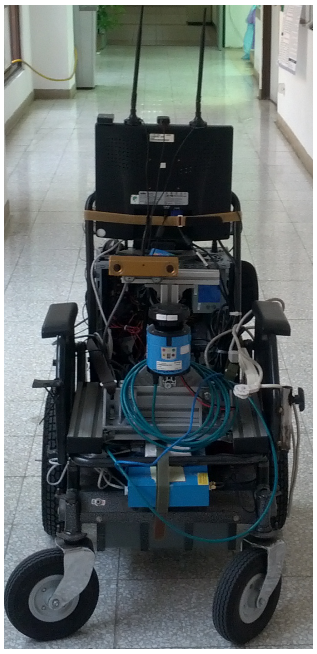
2. The robots are optimized to carry the sensors, the sensors are, however, not yet optimized for all means of transportation. A pedestrian would for example have to carry an unreasonable amount of weight. Optimizing the sensors for mobility lies outside the scope of this thesis.

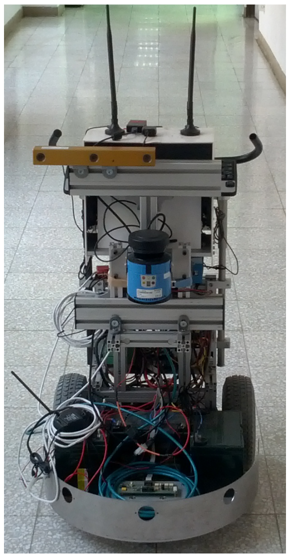
3. The robots used during this project are specifically designed to carry the sensors used in this project. When using a car, either special frames need to be designed to fit the sensors on the car or the sensors need to be modified to fit the car. Installing and adding additional sensors will therefore be more burdensome on a car than on a robot that is designed to carry these sensors.

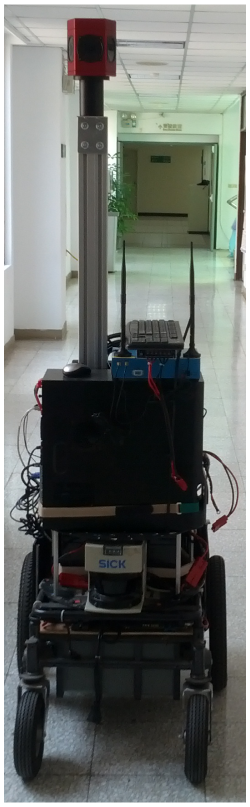
4. It is important to test if the implemented localization and tracking algorithm functions as expected. Since the project is at an early stage, no complex systems are required during this testing phase. Given the previous points robots are therefore a reasonable simplification and are sufficient for testing.

3.1.1 Implementation

Three different robots are used during this project: PAL1, PAL2 and PAL3. These robots carry the sensors that are described in the next section. Metal frames have been designed on which these sensors are installed. Figure 1 shows pictures of these robots.


(a) PAL1


(b) PAL2



(c) PAL3

Figure 1: The three robots

As can be seen in figure 1 the robots do not have the same overall design. However, their sensors, purpose, and functionality are the same. Each robot is thereby equipped with 12 volt lead acid batteries with a DC to AC converter to 110V to supply power for the computer. Since some sensors run on 24 volt, some batteries are coupled together to supply 24 volt. On each robot computational power is provided by a personal computer running Windows XP. The computer is capable of processing sensory data and controls the communication using the DSRC module described in section 3.2. Each robot is thereby equipped with a DC motor for propulsion.

3.2 Sensors

During this project it is assumed that the environment consists of moving objects and traffic users only. In order to determine the current state of the environment sensors are needed. This section introduces the different sensors that were considered during this project: RFID sensors, stereo vision, LIDAR, GPS, accelerometers and inertial measurement units are discussed. Since, as a preprocessing step, some of these sensors are fused, sensor fusion is also shortly explained in this section.

3.2.1 RFID

During the project the use of active RFID was considered as a possible sensor for localization. Active RFID can communicate up to a range of 100 meters and while the signal can be used for communication, it can also be used for localization.

To perform localization, the distance between two traffic users needs to be determined. Huang et. al. [19] use the Received Signal Strength Indicator (RSSI) to measure distance. By measuring the amount with which the broadcasted signal has weakened while traveling between a sender and receiver, the distance between them can be determined. This method, however, has great disadvantages. The signal is sensitive to temperature, humidity, and obstacles. The strength of the signal upon arrival at the receiver's end thus greatly depends on many different factors other than the distance traveled. By using Bayesian estimation an accuracy up to 7 meters can be achieved. This accuracy is however lower than that of a GPS.

By using two RFID tags triangulation can be used by measuring the Angle of Arrival (AOA), for this at least two RFID tags are needed. This setup is displayed in figure 2.

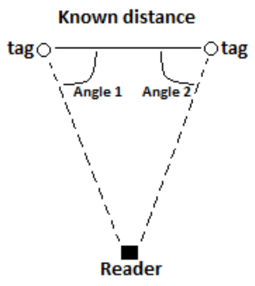


Figure 2: Triangulation

In the figure above the AOA's are labeled as Angle 1 and Angle 2. Using the geometric properties of triangles the distance between the reader and the two RFID tags can be determined. By using two separate RFID tags wrong measurements can be detected when the signal strength received from both RFID tags differs more than a certain threshold. However this setup is still sensitive to environmental influences as described above [8]. Therefore RFID will not be used during the project.

3.2.2 Stereo Vision

By using stereo vision, depth images can be generated. This is done by using two pictures taken by two different cameras that are only installed a few centimeters apart from each other with respect to the horizontal axis. The camera used on PAL1 is the Bumblebee 2. This camera provides a resolution of 1032x776 pixels and its cameras are positioned 12 cm apart from each other. The camera used on the PAL2 is the Bumblebee XB3 and provides a resolution of 1280x960 pixels, the cameras are positioned 12 cm apart from each other. The camera installed on the PAL3 is the ladybug 3 and provides a resolution of 1600x1200 per camera. The ladybug 3 is an omnidirectional camera. All camera's mentioned above are manufactured by Point Grey. Pictures of each of these cameras are given in figure 3.

(a) Bumblebee 2 (b) Bumblebee XB3 (c) Ladybug 3

Figure 3: Stereo Cameras

By using just one of the two images that are generated by a stereo camera, objects such as pedestrians, bicycles, and cars can be detected. After detecting an object the distance to this object can be determined. As an example, to detect pedestrians, a Histogram of Oriented Gradient (HOG) feature extractor can be deployed such as the one readily available from the OpenCV computer vision library. Also, a for pedestrian detection trained SVM classifier is readily available from the OpenCV library. By using an image of one of the cameras, the HOG-feature detector, and the readily trained SVM-classifier, pedestrians can be classified. Other pedestrian detection algorithms are described in [13].

Since the distance measurement with a stereo vision camera is generally less precise than a LIDAR-system, these sensors can be fused as described in section 3.2.6. The LIDAR-system will enhance the distance measurement performance of the stereo vision while the stereo vision system is able of classifying objects given the feature-rich images it produces.

3.2.3 LIDAR

Light Detection and Ranging (LIDAR) is used to obtain relative position measurements between the sensor and objects in the environment. When considering relative localization, a global positioning system (GPS) is less accurate in comparison to a LIDAR-sensor based system and may create noise that translates to a few meters difference from the real position of a traffic user. GPS can therefore generate misleading information upon which the collision risk assessment algorithm can give misleading warnings. It is therefore vital to have precise relative localization between traffic users.

The LIDAR used on the PAL1 and PAL2 is the Sick LMS111 while on the PAL3 the LMS291-S05 is used. The LMS111 has a range of 20 meters at 10% reflectivity while the LMS291 has a range of 30 meters at 10% reflectivity. The error of the LMS111 and the LMS291 are ± 30 mm and ± 35 mm respectively. With better reflectivity the range of both types of laser scanners can go up to about 80 meters. When using a LIDAR sensor for localization and tracking, the data-association problem needs to be solved. This problem consists of associating data points measured at different time-steps as belonging to the same object [6]. A possible solution that tries to solve for the data association problem is given in chapter 4. Pictures of the LIDAR-sensors used during this project are given in figure 4.

(a) LMS111

(b) LMS291

Figure 4: LIDAR sensors

Both sensors discussed in this subsection use laser-light to measure distance. A beam of laser-light is emitted from within the sensor and reflected on the mirror inside the sensor. This mirror can change its angular position along its vertical axis. The mirror of the LMS111 has a range of 270° and can be moved within this range with steps of 0.5°. This means that a maximum of 540 data points can be obtained. The physical setup used in this project only allows for a range of 180° and thus obtaining a maximum of 361 data points. The range of the LMS291-S05 is 180° and also has a resolution of 0.5°.

A known disadvantage of LIDAR is that it performs poorly in rain and snow. However, LIDAR is less expensive to produce than for example a radar system. Radar is therefore more often used than LIDAR in safety warning systems that are available on the market [22]. Disadvantages that can be found in both LIDAR and radar systems are that the ground is measured when a car drives over a bump causing noise in the measurement signal. For this project, using LIDAR is sufficient to conduct experiments and is therefore the cheaper solution. Dealing with noise generated by the LIDAR can be done by using for example Kalman filtering.

3.2.4 Global Positioning System

The Global Positioning System (GPS) module used during this project is the Holux M1000C. This module is used to obtain an inaccurate measurement of the global position of a traffic user. The information provided by the GPS-module is however not accurate enough for classification. Noise in a GPS-signal can cause errors from 3 meters up to 15 meters. The amount of noise is largely dependent on the environment in which the GPS-module is used. Urban areas generally cause more noise because of bad signal reception due to high buildings. A GPS will provide better accuracy in an open field [37].

Global positioning systems that provide sub-meter accuracy do exist. Examples of such systems are Differential GPS (DGPS) and Wide Area Augmentation System (WAAS). However, for DGPS a ground-based station is needed and each GPS module needs to have a clear line of sight with this base station. This is difficult to achieve in an urban environment with high-rise buildings. Also, at this moment WAAS is only available in the United States of America [17]. The GPS-module used in this project can however be used to obtain a first indication of the relative positions of other traffic users and its signal can therefore be used to initiate the Kalman Filter.

3.2.5 Inertial Measurement Unit

All three robots are equipped with an Inertial Measurement Unit (IMU). The IMU is of type 3D-GX1 and is produced by MicroStrain (figure 5). Within this sensor three angular rate gyros, three orthogonal DC accelerometers and three orthogonal magnetometers are combined. This sensor is used to measure acceleration, velocity and rotation of the robots.

When considering short time-intervals the measurements obtained by an IMU are more precise than the velocity and rotation measurements obtained by a GPS. An IMU merely relies on its accelerometer readings to determine acceleration, speed and distance and does not rely on satellite signals. The magnetometer is used to determine rotation. A GPS needs a few measurements to determine rotation and its performance relies heavily on the connection between different satellites. However, over a longer time-interval the error of the IMU accumulates while this is not the case with GPS, thereby, disturbances in the magnetic field can cause the magnetometers to give false readings.

When the information from the LIDAR and the IMU are combined, a more precise reading can be obtained about the relative positions between traffic users. Both the measurements obtained by the IMU and the LIDAR can be used as an input to the Kalman filter.

Figure 5: Inertial Measurement Unit

3.2.6 Sensor Fusion

As explained previously, images from a stereo camera are used for pedestrian detection while LIDAR is used for distance measurements. The stereo camera is able to give a depth indication about the pedestrians it detects. By overlaying the distance measurements of the LIDAR and the stereo camera LIDAR data can be merged with data of the camera. Using this method, pedestrians can be classified on the LIDAR image.

The LIDAR has a higher precision than the stereo camera and will therefore give more reliable results while tracking. By detecting pedestrians a different motion model can be selected for Kalman filtering. The pedestrian detection therefore helps improving the results of a tracking system. An example of a result of a fusion between the stereo camera and the LIDAR can be seen in figure 6.

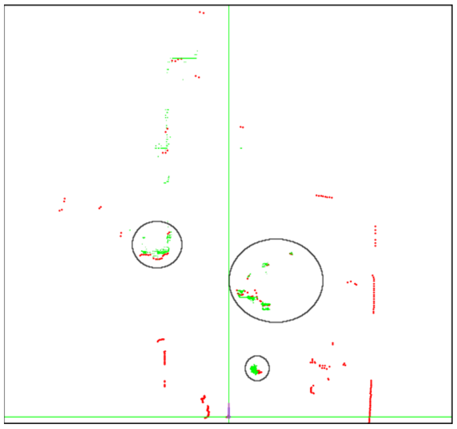


Figure 6: Sensor Fusion souce: [12]

In the figure above the purple dot in the bottom-middle of the figure represents one of the robots described in section 3.1, the red dots represent distance measurements obtained by the LIDAR while the green dots represent the data obtained by the stereo camera. The circled areas represent pedestrians detected by the OpenCV pedestrian detector using the camera images [12].

This section however only shows how the camera and LIDAR are used in order to detect pedestrians. In a similar way cars and possible other classes can be detected as well. At this moment the functionality of the pedestrian detection is not yet put into use and will not further be described in this thesis.

3.3 Communication

Different traffic users can detect different moving objects. Using communication, these detected moving objects along with the traffic user's own location and rotation can be shared with other traffic users. This enables traffic users to notice moving objects that they have not detected themselves. Communication between traffic users thus increases the range at which moving objects can be detected. The safety warning system builds a map containing all moving objects detected by the traffic user itself and the ones it retrieved from other traffic users, how this map is built is explained in the next chapter.

The safety warning system uses the collision risk assessment algorithm to determine if a warning should be displayed. In the current implementation, this algorithm only uses simulated data. It will, however, in the near future use the previously mentioned map to extract features. The precision of the algorithm will thus depend on the completeness of this map and is therefore also dependent on the communication between traffic users. Hence, it is important that data is reliably shared. This section describes two different data sharing algorithms that both implement a VANET (Vehicular Ad-Hoc Network) and the communication hardware that is used. Section 3.3.1 describes this hardware, the next section elaborates on the actual implemented approach using single-hop broadcasting. Another approach using multi-hop multicasting is described in section 3.3.3.

3.3.1 Hardware

Dedicated Short Range Communications (DSRC) modules are designed specifically for automotive applications [31]. DSRC modules are preferred over other communication systems because of several reasons [44], a few of these are listed below:

- DRSC operates on a licensed frequency band.
- Supports high speed and low latency.
- Works with vehicles that are driving at high velocity.
- The performance is immune to extreme weather conditions.
- Specifically designed for safety applications.

- Preferred over Wi-Fi because of the proliferation of Wi-Fi devices.

The DSRC modules used in this project are of type IWCU 3.0 OBE and are manufactured by the Industrial Technology Research Institute of Taiwan. These modules are able to communicate over distances of up to 100 meters where most Wi-Fi hardware only has a range of about 30 meters. When driving at 120 km/h and applying the 2 second rule explained in the introduction the distance between two cars becomes approximately 66 meters. The modules thus have a range large enough to support this distance. The modules communicate to the computer installed on the robot using LAN and the UDP protocol. A picture of the DSRC module is given in figure 7.

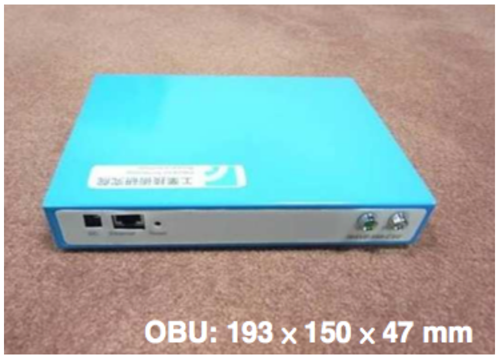


Figure 7: The DSRC module

3.3.2 Broadcasting Algorithm

When implementing a communication algorithm a few criteria should be kept in mind. Some of these criteria are the load on the network that the algorithm imposes, the speed at which packages are delivered and the reliability of the network. When using a multi-hop multicast network, messages might reach farther than when using a single-hop broadcasting algorithm. It is, however, difficult to determine until what physical range the messages should be forwarded and messages might take several hops to reach a destination that could have also been reached in one hop. This causes unnecessary time delays in a system in which time delays can be critical. However, multi-hop can be useful when there is no direct line of sight causing the network to not be able to reach certain traffic users [33].

This section describes a one-hop broadcasting algorithm which only sends messages to traffic users within one hop range (i.e. the range of the DSRC module) as shown in figure 8

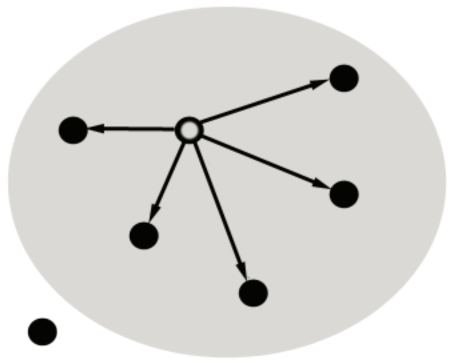


Figure 8: Single-hop Broadcasting

In figure 8 the source is given by an open circle while the black circles represent traffic users. The grey circle represents the range of the DSRC module which is about 100 meters when having direct line of sight. The traffic user that is broadcasting in figure 8 can therefore not reach all traffic users visible in the figure.

Implementation
The broadcasting algorithm implemented during this project uses the Windows sockets (Winsock) API. The implementation is divided in a client side and a server side. The client side sends information to other traffic users while the server side receives information from other traffic users. The client side consists of three methods: **OpenConnection()**, **send()** and **closeConnection()**.

The method **OpenConnection** opens a connection to a server by setting its IP address and listening port. The method furthermore specifies that the socket type to be used is a datagram socket, which means that the user datagram protocol (UDP) is used. The user datagram protocol is chosen over the transmission control protocol (TCP) mainly because UPD is a lot faster than TCP since no acknowledgement messages need to be sent out causing the overhead of the protocol to be much smaller.

The method **send** uses the connection that was opened by the **OpenConnection** method to send packages over the network. As explained in the beginning of this section a message contains the position of a traffic user and possible other data. The structure in which this data is stored first needs to be converted into a character array before it is sent out over the network. When the message is received at the server side it will be converted back into its original type. The method **closeConnection()** closes the socket.

Experiments and Results
Four different experiments were conducted using the robots explained in section 3.1 and the single-hop broadcasting algorithm. During these experiments the

DSRC modules of the robots explained in section 3.1 are broadcasting messages at the same time. All four experiments were conducted in a laboratory. These four experiments are listed below:

1. Each robot sends out 10 packages per second at the same time during a period of 30 minutes.
2. Each robot sends out 20 packages per second at the same time during a period of 5.4 minutes.
3. Each robot sends out 20 packages per second at the same time during a period of 7.35 minutes.
4. Each robot sends out 20 packages per second at the same time during a period of 10.8 minutes.

In the first experiment an average of 0.9992 percent of the packages were received by all robots, while in the second experiment only 0.9157 percent of the packages were received. In the third experiment 0.9975 percent of the packages were received while in the last experiment 0.9980 percent of the packages were received. In the second experiment the robots had a bigger distance to each other and one robot was placed outside of the laboratory, causing the signal to weaken and therefore causing a higher loss of messages. In the single-hop broadcasting algorithm the DSRC modules will start broadcasting over one channel and to one ID. This results in package collisions in the wireless network causing the transfer rate to drop. The more traffic users are broadcasting, the higher the message collision rate will become. It is expected that the algorithm will work when it is applied in a real-world application but that it will fail once the amount of traffic users grows too large. The next section describes a protocol in which traffic users are only allowed to send to a select group of other traffic users by using a tree. This protocol is expected to have a lower collision rate and a larger range.

3.3.3 Discussion: Multiple Initiators Flooding Algorithm

This section discusses a multi-hop multiple initiators flooding algorithm that can extend the range of a normal single-hop broadcasting algorithm and poses less load on the network. Where the message complexity of the single-hop broadcasting is $n - 1$ (the initiator does not receive a message from itself) the message complexity of a flooding algorithm is $2l - (n - 1)$ where l represents the number of edges in the graph where l could take the value of n^2 in a worst case scenario.

In figure 9 a flooding tree can be seen. The open circle represents the root of the tree. As can be seen in the picture, because there are multiple-hops possible in the algorithm, the nodes (traffic users) that are not in direct reach of the root of the tree can still be reached.

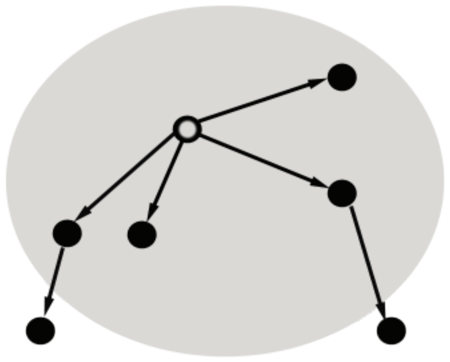


Figure 9: A flooding tree

The multiple initiators flooding tree building algorithm consists of two parts: the tree initiation part and the tree building algorithm. Both algorithms run on all nodes i. However, the tree building algorithm is only called when a connection breaks or a new connection is made. Every time a connection between two traffic users breaks or a new connection is available a new tree needs to be built.

Due to this requirement a few changes have to be made to the normal flooding algorithm. Every node contains a variable *my_root* in which it stores the root of the communication tree, each node thereby possesses an ID and a pseudo-id called TID (tree-ID). Messages sent out over the network are denoted with $< M >$.

When initiating a tree each message will contain the TID of the node who is initiating a new network, this is denoted as $< M(TID) >$. The ID of a node never changes while the TID does change when the topology of the network changes. Such a change occurs when a traffic user either leaves or enters a network.

When building a new tree, upon receiving message $< M(ID) >$ a node will check if its current root ID variable *my_root* is higher than the ID stored in the message ($ID <$ *my_root*). When the ID in the message is higher than *my_root*, the new ID is accepted as the root ID and *my_root* is set to this ID. When the ID is not higher the node will respond accordingly (further explained in the algorithms below). In a changing topology, however, there exists a chance that the root leaves the tree (e.g. when the root drives faster than other nodes). This causes all nodes that detect this change to initiate a new tree. In a normal flooding algorithm where no (variable) TID is available this poses a problem since the statement $ID >$ *my_root* can never be true anymore and a new tree cannot be built. By using the TID this problem is solved. Every time a node leaves the tree each node that detects this change increases its TID with the amount of nodes available until their IDs are higher than the previous root ID. Since, in the beginning of execution, all TIDs are based on the unique ID of each node, increasing the TID with the amount of participants always generates a new unique TID. The algorithms implementing this idea are given below.

```
While(true)
{
  Check if new connections are available or if already existing
      connections are lost;
  if( connection lost || new connection available)
  {
    Increase TID with amount of participants until TID > $my\_root$
    Send message <M> containing ID and TID.
  }
}
```

Algorithm 1: Tree Initiation Algorithm

```
if(received(<M(TID)> from $b_j$)
{
  if($my\_root < M(TID)$)
  {
    send <PARENT> to $b_j$
    $my\_root := TID$
    forward <M> to all neighbors except parent.
  }
  else if($my\_root == M(TID)$ //Node i is the root itself.
  {
    send <ALREADY, TID> back to $b_j$
  }
  else if($my\_root > M(TID)$)
  {
    send <ALREADY, TID> back to $b_j$
  }
}
```

Algorithm 2: Tree Building Algorithm

In order to explain the algorithms given above, traffic users are denoted as processes. Each process starts with sending out a message. Whenever a process receives a message, it checks if the ID in the message is either higher, equal, or lower. When the TID is higher, the node sets the variable *my_root* to the TID in the message.

When a process receives a message with a TID that is equal to the variable *my_root* the process is the root itself. While initiating a new tree all processes are still allowed to send messages to all other processes. When process a and b are both connected to each other but also to the root it is possible for the root to receive a message containing its own TID. This can happen when the root (process a) sent a message containing its TID to both b and c and when the message arrived earlier at process b than at process c. Process b forwards the message it got from the root to process c. When the message from process b arrives earlier at process c than the message sent by the root, process c considers process b as its parent and sends out a message to all processes that are not his parent. Since process a is not the parent of process c, process c sends a message to process a not knowing that this is the actual root. The root can therefore receive a message containing its own TID. A situation in which this happens occurs when the delay between the root and process c is higher than the total delay over the path: root, process b, process c.

The root will then send a message back to node c containing $< ALREADY, TID >$. If the variable *my_root* is higher than the TID in the message node i will send a message back also containing $< ALREADY, TID >$.

Problems

Even though the multiple initiators flooding algorithm proposed here can deal with changing topologies, a few problems still are to be solved. Using only the three robots described in section 3.1 no problems will occur. However, given the (usually) large scale at which intelligent transportation systems are deployed, this algorithm will have to deal with a large amount of participants. Traffic users will constantly be in and out of reach of different networks and networks can grow very big in physical range. To overcome the problem of large networks, a GPS range can be defined around the root.

Also, to overcome the problem of constantly having to build a new tree a time limit can be posed onto a network during which a network has to exist. The biggest problem however consists of assigning a unique TID to each traffic user as very large numbers will have to be used due to the immense amount of traffic users throughout the world. Before a solution to this problem is found this algorithm can only be used in small ITS applications where the amount of traffic users is known.

Chapter 4

4 Localization & Tracking

In order to predict events that may follow from a certain traffic situation, localization and tracking of moving objects needs to be performed. The localization and tracking part uses the measurements that are obtained by the sensors, described in the previous chapter, to compose a map with relative distances and orientations between moving objects. The more accurate the localization, the better the predictions become. Therefore, in a safety warning system precise (relative) localization is a must. When it is possible to communicate the exact global location of a traffic user to other traffic users, the localization and tracking problem becomes a much easier one to solve.

As explained in subsection 3.2.4, global positioning systems that perform global localization with sub-meter accuracy do exist but are either very expensive, are field specific or their service is limited to a select few countries. Having the precise global position of each traffic user is useful, but not necessary, and for this application almost impossible. For all of the described global positioning systems, a direct line of sight with for example a base station is required. This is however often impossible in an urban environment. Since only the relative positions between traffic users and obstacles are of importance other (relative) localization methods can be used.

By measuring the distances and angles between different objects relative localization can be performed. A wide variety of sensors can be used to obtain these measurements. In order to associate different measurements that belong to the same object with each other, the data association problem needs to be solved. To measure the relative distance the LIDAR sensors described in section 3.2.3 are used. This section describes the implementation of the localization and tracking part using the LIDAR. First the pose estimation calculated from LIDAR data is explained, then the tracking algorithm with which moving objects are tracked is explained. The Kalman filter performing state updates is explained in section 4.3 while at last the multiple hypotheses tracking algorithm is elaborated on.

4.1 Pose Estimation

Pose estimation is necessary to compute the ego-motion of a traffic user. The pose of a traffic user is computed using LIDAR data and is therefore relative. A pose (w, T) consists of the rotation w and translation T with respect to the previous pose. The initial pose is set to (0,0). Because error accumulates over time, poses become increasingly noisy over time. This noise can partly be corrected for by using a Kalman filter. To keep track of the pose of a traffic user, consecutive LIDAR-scans are aligned. This process is repeated every time a new LIDAR-scan is obtained. When the traffic user is moving each LIDAR-scan is slightly different from the previous one, due to sensor noise this can also be the case when the traffic user is not moving. By aligning two scans obtained at two consecutive time steps the translation and rotation of the traffic user can be computed between these two time steps. The amount a scan of the current time-step has to be translated and rotated to fit the scan of the previous time-step equals the rotation and translation of the traffic user.

In [28], Lu & Millios explain the Dual Correspondence algorithm. This algorithm tries to solve for the translation and rotation between two point clouds (in this case the LIDAR-scans) given an initial guess. In this algorithm, a normal iterative closest point (nearest neighbor) method is combined with the Matching-Range-Point Rule method. Both methods are explained below. In the Dual Correspondence algorithms a scan S_{ref} and S_{new} are required. S_{ref} corresponds to the scan made in the previous time step while S_{new} corresponds to the scan obtained in the current time step.

In the closest point rule (Iterative Closest Point: ICP), for each point in S_{new} the point with the smallest euclidean distance in S_{ref} is assigned as the corresponding point. In the Matching-Range-Point Rule an angular range B_w around a point in S_{new} is determined. An example is given in figure 10. In this figure, θ represents the angle of the mirror of the LIDAR-sensor. Within the range B_w, the point P' on S_{ref} that has the lowest length (radius) difference to point P on S_{new} will be chosen as the corresponding point of P. Hence P' does not necessarily have to be the closest point to P. At the end of one iteration of the Dual Correspondence algorithm, point pairs (P, P') are obtained from both the closest point rule and the Matching-Range-Point Rule. For both methods thresholds can be set to determine the maximum distance in the closest point rule or length difference in the Matching-Range-Point rule.

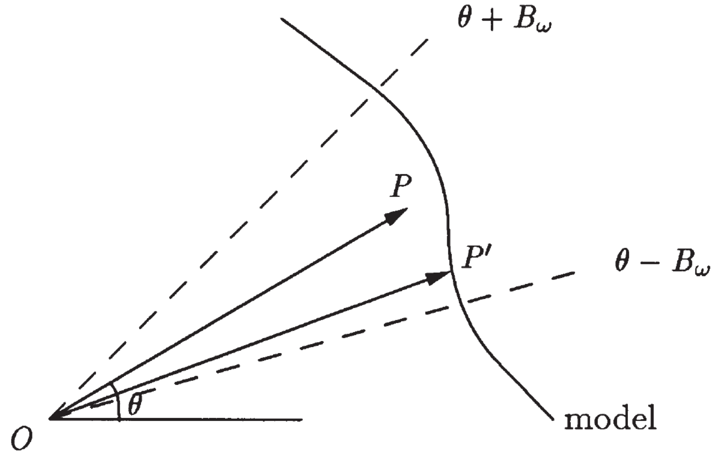


Figure 10: Angular range, Source: [28]

By computing the least squares estimates (linear regression) using the point pairs, the angle and the translation (w, T) can be determined for both the nearest neighbor method and the Matching-Range-Point Rule method. The angle w is obtained from the Matching-Range-Point Rule method while the translation is obtained from the closest point method. Then all points in S_{ref} are transposed using the formulas given in equation 1 and 2.

$$P'_x = \cos(\omega) * P_x - \sin(\omega) * P_y + T_x \tag{1}$$
$$P'_y = \sin(\omega) * P_x + \cos(\omega) * P_y + T_y \tag{2}$$

After transposing the points the algorithm is repeated again for a certain amount of iterations. The algorithm is well known to get stuck in local minima, to overcome this problem an initial guess can be provided to the algorithm. This initial guess can for example consist of a translation and rotation (w, T) that lies close to the solution. In this implementation the outcome of the previous time-step is used as the initial guess. The dual correspondence algorithm is given below.

```
while(condition)
{
  Apply closest point rule;
  Apply matching-range-point rule;
  Compute the least squares estimates (w1, T1) of the closest
      point rule;
  Compute the least squares estimates (w2, T2) of the Matching-
      Range-Point rule;
  Use w2 and T1 as the solution;
  Transpose the points of $S_{ref}$;
}
```

Algorithm 3: Dual Correspondence Algorithm.

The *condition* variable in the while loop can be replaced by different conditions. The algorithm can for example terminate when a certain least squares estimate threshold has been reached. In this project the algorithm terminates after a fixed amount of 20 iterations. The algorithm converges rather fast and using a fixed amount of iterations forces the algorithm to always terminate after an invariable amount of time. This increases time performance and ensures that the algorithm always returns a result.

In figure 11 a screenshot is shown in which data points from different consecutive laser scans are aligned. The red line shows the trajectory that the traffic user made through the environment. This trajectory is determined using the dual correspondence algorithm.

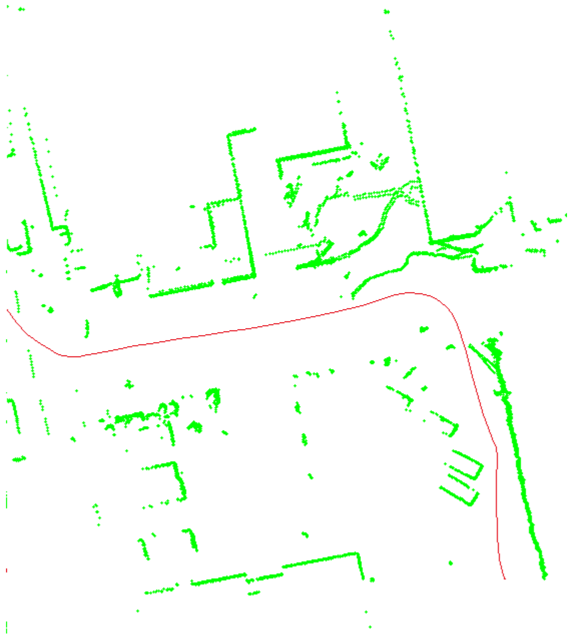


Figure 11: A map created using consecutive laser scans. The distance from left to right and up to down is about 50 meters and 400 scans were used. The length of the trajectory is about 90 meters.

The dual correspondence algorithm described here uses the current laser scan as S_{new} and the laser scan obtained five scans before as S_{ref}. The difference in translation and orientation between two consecutive scans is almost negligible. Using consecutive scans in combination with a poor initial guess can cause the algorithm to get stuck in a local minima during the first few iterations while having a relative high error. A larger separation between scans is therefore desirable. However, when using two scans that are separated by a too large time-interval, the two scans will show no resemblance with each other and no solution can be found. It was empirically determined that in this project a difference of 5 scans results in the least error. Depending on the quality of the initial guess this difference can be smaller.

A distance threshold is used to determine if a point pair is valid. When the two closest points P and P' lie too far away from each other they will not be associated with each other to avoid divergence of the algorithm. As the algorithm converges, the distances between correspondence points become smaller, the threshold should therefore also become smaller. During the first iteration this threshold is set to 9500 millimeter and is decreased every iteration using the following formula: $threshold = threshold - (9500/max_iterations)$, where $max_iterations$ is the total amount of iterations that the algorithm is iterated for [41]. The threshold was determined empirically, when using a bigger threshold the algorithm starts to diverge. The largest possible threshold still yielding convergence of the algorithm is desirable since this allows for the biggest distance margin between point pairs. This method increases the performance of the algorithm. The pose obtained from this algorithm is fed into a Kalman filter using a motion model which is explained in section 4.3.

4.2 Moving Object Detection & Tracking

When a traffic user detects a moving object a few processes are executed: the moving objects are shared with other traffic users and the multiple hypotheses tracking algorithm instantiates multiple Kalman filters and chooses the most likely hypothesis. This section explains how moving object detection is performed and how moving objects are tracked. When using LIDAR-data to perform tracking, first the pose of the traffic user itself needs to be computed, this was explained in section 4.1. Then, scans of two consecutive time-steps need to be segmented, the centroids of each segment need to be determined and data-association needs to be performed in order to determine which centroid of the current LIDAR-scan belongs to what centroid in the previous LIDAR-scan. These operations are performed every time a new LIDAR-scan is obtained. This procedure is described by both [39] and [50]. This section first explains how scans are segmented, it then continues with explaining how data association is performed, at last it explains how tracks are initiated and maintained.

4.2.1 Segmentation

When performing data association, segmentation of the laser scans is a necessary preprocessing step. Depending on the LIDAR scanner each scan consists of 361 or 540 data points. During segmentation, data points that lie close to each other are grouped together. The segmentation algorithm used in this project is described in [9]. In this algorithm breakpoints are detected using a dynamic threshold. If the euclidean distance between two points p_n and p_{n-1} is bigger than some threshold D_{max} ($||p_n - p_{n-1}|| > D_{max}$) then both points are classified as breakpoints. In this case k_n^b and k_{n-1}^b are set to true, k_n^b denotes the indices at which breakpoints are found and therefore functions as a flag. The superscripted b denotes that it is a flag for breakpoints and $1 \leq n \leq N$ denotes the amount of data points. The variable k_n^r represents a rupture flag and is set to TRUE at the indices of points that are most likely invalid measurements. Since invalid measurements are already filtered out before the segmentation process, the rupture flag will not be used.

A virtual line with an angle λ is defined with respect to the scanning direction Φ_{n-1}. This line indicates under what angle environmental lines (e.g. a wall) are still reliably detectable. The variable $\Delta\Phi$ corresponds to the angular resolution of the laser scanner (0.5 degrees using the scanners described in section 3.2.3). Figure 12 gives an intuition of how these angles are used.

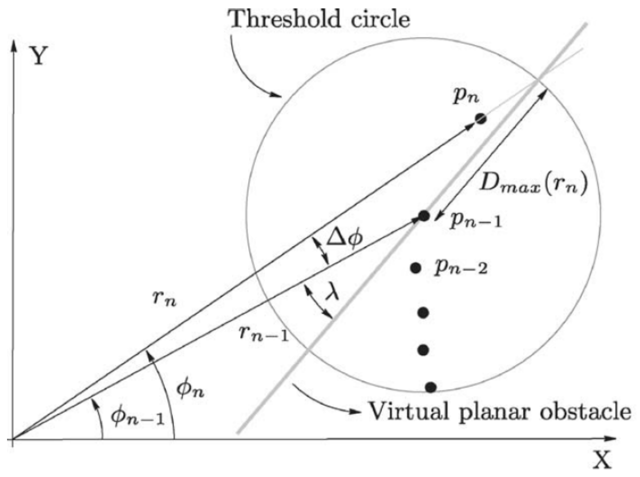


Figure 12: Segmentation, source: [9]

In each step the threshold D_{max} is computed depending on the radius r_n that a point p_n has with respect to the LIDAR-scanner. By using the virtual line under an angle of λ, the biggest range for point p_n can be extrapolated. Since the hypothetical range distance r_n^h for the n^{th} point is related to the radius of the previous point r_{n-1} by $r_{n-1} \cdot \sin\lambda = r_n^h \cdot \sin\lambda - \Delta_\Phi$ it follows that the biggest acceptable range between two points is given by

$$||p_n^h - p_{n-1}|| = r_{n-1} \cdot \frac{\sin\Delta_\Phi}{\sin(\lambda - \Delta_\Phi)} \tag{3}$$

Since noise should also be taken into account the threshold will become

$$D_{max} = ||p_n^h - p_{n-1}|| + 3\sigma_r \tag{4}$$

where σ_r is the standard deviation of the measured distances. The algorithm is given below.

```
$k_1^b = FALSE$
for(n = 2; n < N; n++)
{
   $D_{max} = ||p_n^h - p_{n-1}|| + 3\sigma_r$
   if(||$p_n^h - p_{n-1}$|| > $D_{max}$)
   {
      $k_n^b = TRUE$
      $k_{n-1}^b = TRUE$
   }
   else
   {
      $k_n^b = FALSE$
   }
}
```

Algorithm 4: Breakpoint Detection

After the different segments have been determined, the ones that contain too few data points are discarded. For each remaining segment, the segment centroid is determined by simply calculating the average of the x and y positions of all points in the segment. These segments are then used during the data association step described in the next section.

4.2.2 Data Association

Data association is necessary when landmarks cannot uniquely be identified by using only the measurement data as is the case with LIDAR-data. Due to random sensor noise, clutter and other interfering targets in the LIDAR-data data association is a non-trivial problem [46]. The data association process tries to associate measurements made in different time steps belonging to the same moving object. When a measurement cannot be associated with an existing object, a new moving object has been detected. One measurement can only be associated to one moving object at a time. In this project, the data association problem is solved by using segmentation, ICP and a nearest neighbor algorithm.

Each new scan S_{new} is segmented using the algorithm described in the previous subsection. Each segment found in the scan S_{new} is then matched to the complete previous scan S_{ref} using ICP as described in section 4.1. Each segment for which there is a solution will therefore be matched to the segment in the reference scan S_{ref} that is most likely representing the same object. In this way the orientation of each segment and the translation of that segment relative to the previous scan is determined. Figure 13 shows the first iteration during the ICP process between a segment (in this case a moving car) and the previous scan.

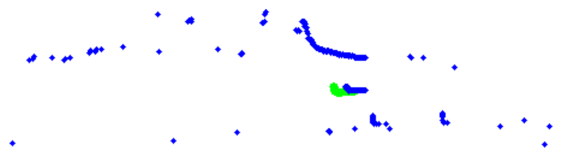


Figure 13: ICP on segments (first iteration)

In figure 13 the segment obtained from scan S_{new} is displayed in green while the reference scan S_{ref} is displayed in blue. Figure 14 shows the last iteration of the ICP process. As can be seen the segment is now aligned.

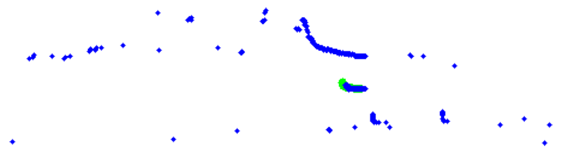


Figure 14: ICP on segments (last iteration)

As explained in subsection 4.2.1, before the ICP algorithm is executed the scan S_{ref} and S_{new} are segmented. Then the centroids of each segment from both scans S_{ref} and S_{new} are computed. The centroids obtained for scan S_{new} are then translated using the translation and orientation obtained during the ICP process. In this way the centroids that are most likely to belong to the same segment and therefore to the same object are (almost) superimposed.

By using the nearest neighbor algorithm every centroid in the scan S_{new} is associated with the centroid in S_{ref} to which it has the smallest euclidean distance. Also thresholding is used to determine if the distance between the two closest centroids is too large for the centroids to belong to each other, in this case the two centroids are farther apart than 500cm and are discarded. For some segments the ICP algorithm diverges and does therefore not give a feasible solution. This is normally the case when no corresponding segment can be found in the reference scan to which the segment in the new segment can be mapped. These segments are discarded.

In the next step S_{new} will be used as the reference scan S_{ref} and the at that time current scan will then become S_{new}. It is therefore not necessary to compute the segments and centroids of S_{ref} since they were already computed in the previous time step when S_{ref} still was the new scan S_{new}. Only at initiation do the segments and the centroids of S_{ref} have to be computed.

4.2.3 Track Initiation

As explained in the previous section centroids of different segments that represent the same object are associated with each other. When two centroids are associated with the same object, depending on the centroid in the reference scan a track is either updated or initiated. When a centroid of the reference scan has not yet been assigned to a track a new track is initiated. Both the centroid in S_{ref} and the matching centroid in S_{new} are assigned an id. A track therefore consists of the centroids belonging to the same object (segment) in different scans made at consecutive time steps and each track is thereby assigned an id.

Using the previous pose results from the ICP algorithm an accumulated pose can be computed. The positions of centroids in a laser scan are mapped using this accumulated pose. Centroids belonging to a track of a non-moving object are likely to stay at almost the same global location in each consecutive scan (given that there is no noise). However, because of noise, and because of the movements of the traffic user, segments of the same object tend to look a bit different in each consecutive laser scan and segment centroids therefore also change location.

After an object is visible for twenty consecutive frames and the distance traveled by this object is greater than a certain threshold, the segment is considered to be a moving object. The threshold used during this project is 4 meters. Each moving object is assigned a label and its path is stored in a vector structure. The result of the moving object detection can be seen in figure 15. The red line shows the path made by the car while the blue line shows a moving object (a car in this case).

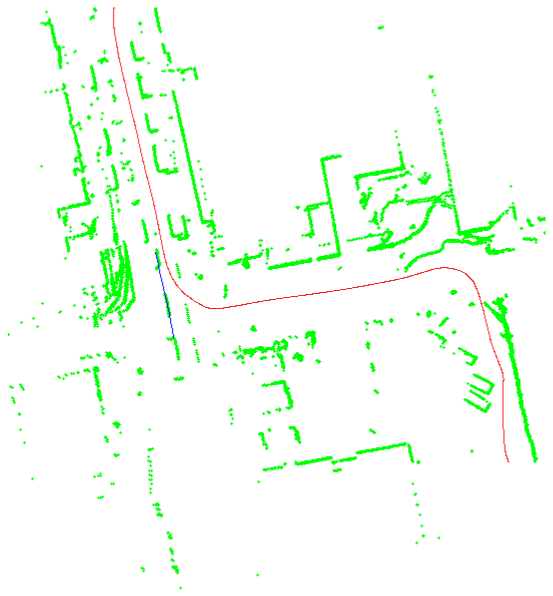


Figure 15: Moving Object Detection: the red line is the pose of the traffic user while the blue line is a detected moving object. The distance from left to right is about 50 meters while the distance from up to down is about 90 meters. About 600 scans were used.

4.3 The Extended Kalman Filter

To perform localization and tracking of the traffic user itself, the other traffic users, and moving objects, an augmented-state Extended Kalman Filter (EKF) [23] is deployed. In an augmented-state EKF all states of all traffic users and moving objects are augmented into one state vector, which is variable in size. Depending on the amount of moving objects and traffic users at each point in time, the state vector changes in size.

First the basics of Kalman filtering are explained in subsection 4.3.1. The state vector used in this project is explained in subsection 4.3.2. The motion model used to model the movements of traffic users is explained in subsection 4.3.3. The constant velocity model used to model movements made by moving objects that are not traffic users or not yet classified as such, is explained in subsection 4.3.4. At last the measurement model is explained in subsection 4.3.5.

4.3.1 The Algorithm

A Kalman filter [24] recursively estimates variable values at each time step using sensor measurements, control inputs and a prediction model. The variables are usually grouped together to form a state vector μ. The control input u often consists of the input given to the actuators of a system (e.g. the amount of throttle in an airplane). The previous state μ_{t-1} and the current control vector u_t are the inputs of the prediction model which generates a prediction of the current state $\bar{\mu}_t$. By using a weighted average K, the predicted state $\bar{\mu}_t$ and the measured state are combined and form the estimated current state μ_t. The weights are computed using the predicted covariance $\bar{\Sigma}$. The predicted covariance is dependent on the prediction model G, the process noise Q, the observation model H, and the observation noise R. The observation model maps the predicted state variables into measurement space. The weighted average is given by the Kalman gain K.

A Kalman filter is, however, not capable of dealing with non-linear input. Therefore, an extended Kalman filter is deployed that performs linearization using the Jacobians of the prediction model and the observation model to generate the predicted covariance. In this way, non-linear prediction and observation models (often functions) can be used, their Jacobians bear the variables G and H respectivily. A Jacobian matrix contains the first order partial derivatives of each function within a matrix with respect to each variable apparent in the matrix. The extended Kalman filter algorithm is given below.

```
$\bar{\mu}_t = g(\mu_{t-1}, u_t)$
$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + Q_t$
for all observed features by each traffic user
{
    $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + R_t)^{-1}$
    $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
    $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
}
return $\mu_t$, $\Sigma_t$
```

Algorithm 5: EKF Algorithm

In the algorithm above, the function g represents the state transition function. This function computes the predicted state $\bar{\mu}_t$ and has input variables μ_{t-1} and u_t. The function h maps the predicted state $\bar{\mu}_t$ to the measurement space. The variable z_t represents the measurements obtained at time t. The covariance matrix is given by Σ and is used to determine the predicted covariance matrix $\bar{\Sigma}$. This matrix is then used to compute the Kalman gain K. In this project two prediction models were used: to predict the states of traffic users a motion model is used, while the states of moving objects are predicted using a constant velocity model.

4.3.2 State Vector

In an augmented-state extended Kalman filter the states of all traffic users and all moving objects are augmented into one larger state vector. The state vector of a traffic user i at time t is given by $T_t^i = \begin{bmatrix} x_t^i & y_t^i & \theta_t^i \end{bmatrix}$ and contains the x and y positions and the rotation θ of that traffic user. The state vector of a moving object j at time t is given by $O_t^j = \begin{bmatrix} x_t^j & y_t^j & vx_t^j & vy_t^j \end{bmatrix}$ and consists of the x and y positions and the velocity in both the x and y direction, relative to a traffic user i. The x and y positions and the rotation θ of a traffic user are relative to the initial position of that traffic user as explained in section 4.1. Since multiple moving objects can be detected by a traffic user, multiple state vectors O can exist. Also, the state vector T and all state vectors O of a traffic user are shared to all other traffic users within range of communication. In the end, state vectors T_t^i and O_t^j with $1 \leq i \leq N$ and $1 \leq j \leq M$ are obtained where N represents the amount of traffic users and M the amount of moving objects. All these state vectors are merged into one bigger state vector X as shown in the formula below:

$$X_t = \begin{bmatrix} (T_t^1)^T & \dots & (T_t^N)^T & (O_t^1)^T & \dots & (O_t^M)^T \end{bmatrix} \tag{5}$$

4.3.3 Motion Model

The motion model is used to predict the position and rotation of a traffic user. It thus predicts the values for state vector T for each traffic user and stores these values in the predicted state vector $\bar{T}_t^i$. This predicted state vector is computed using the formula below.

$$\bar{T}_t^i = g_T(T_{t-1}^i, u_t^i) + w_T^i \tag{6}$$

where $w_T \sim \mathcal{N}(0, Q_T^i)$ models the Gaussian process noise. The function $g_T(T_{t-1}^i, u_t^i)$ is represented by the formula below:

$$\begin{bmatrix} x_t^i \\ y_t^i \\ \theta_t^i \end{bmatrix} = \begin{bmatrix} x_{t-1}^i \\ y_{t-1}^i \\ \theta_{t-1}^i \end{bmatrix} + \begin{bmatrix} \cos(\theta_{t-1}^i) & -\sin(\theta_{t-1}^i) & 0 \\ \sin(\theta_{t-1}^i) & \cos(\theta_{t-1}^i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \triangle x^i \\ \triangle y^i \\ \triangle \theta^i \end{bmatrix} \tag{7}$$

The vector $\begin{bmatrix} \triangle x^i & \triangle y^i & \triangle \theta^i \end{bmatrix}^T$ represents the control vector u and its values are obtained using the inertial measurement unit. The Jacobian $G_{T_t}^i$ of the function g_T is given by the formula $G_{T_t}^i = \frac{\delta g_T(T_{t-1}^i, o_t^i)}{\delta T_t^i}$ which results in the Jacobian matrix below:

$$G_{T_t}^i = \begin{bmatrix} 1 & 0 & -\sin(\theta_{t-1}^i) \triangle x^i - \cos(\theta_{t-1}^i \triangle y^i) \\ 0 & 1 & \cos(\theta_{t-1}^i) \triangle x^i - \sin(\theta_{t-1}^i \triangle y^i) \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

In this project the matrix Q_T^i is initialized to:

$$Q_T^i = \begin{bmatrix} 10000 & 0 & 0 \\ 0 & 10000 & 0 \\ 0 & & 0.025 \end{bmatrix}$$

4.3.4 Constant Velocity Model

The constant velocity model predicts the states of the moving objects $\bar{O}_t^j$ using the following formula:

$$\bar{O}_t^j = G_{O_t}^j \cdot O_{t-1}^j + w_O^j \tag{9}$$

In the above formula the Gaussian process noise is modeled by $w_O \sim \mathcal{N}(0, Q_O^i)$. The variable $G_{O_t}^j$ represents the matrix:

$$G_{O_t}^j = \begin{bmatrix} 1 & 0 & \triangle t & 0 \\ 0 & 1 & 0 & \triangle t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

By multiplying the above matrix with the state vector O_t^j a new predicted state is obtained. Since the matrix G and state vector O are linear no Jacobian is needed. Both the Jacobian of the motion model $G_{T_t}^i$ and the Jacobian of the constant velocity model $G_{O_t}^j$ are merged in the bigger matrix G as shown below.

$$G_t = \begin{bmatrix} G_{\mu_t}^1 & 0 & \dots & \dots & \dots & 0 \\ 0 & \ddots & \ddots & & & 0 \\ \vdots & \ddots & G_{\mu_t}^N & \ddots & & 0 \\ \vdots & & \ddots & G_{O_t}^1 & \ddots & 0 \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & \dots & 0 & G_{O_t}^M \end{bmatrix} \tag{11}$$

In this project the matrix Q_O^i is initialized to:

$$Q_O^i = \begin{bmatrix} \sqrt{5000} & 0 \\ 0 & \sqrt{5000} \end{bmatrix}$$

4.3.5 Measurement Model

A measurement consists of the relative position between a traffic user i and a moving object. When a moving object is associated with moving object j apparent in the state vector the measurement model is given by:

$$z_{T^i}^{O^j} = h_{TO}(\bar{T}_t^i, \bar{O}_t^j) + w_{TO_t} \tag{12}$$

In the above formula the observation noise is modeled by $w_{TO} \sim \mathcal{N}(0, R_O^i)$ and the function $h_{TO}(\bar{T}_t^i, \bar{O}_t^j)$ is given by the formula below.

$$h_{TO}(\bar{T}_t^i, \bar{O}_t^j) = \begin{bmatrix} \sqrt{q} \\ atan2(\bar{y}_t^j - \bar{y}_t^i, \bar{x}_t^j - \bar{x}_t^i) \end{bmatrix} \tag{13}$$

In the formula above q represents $(\bar{x}_t^j - \bar{x}_t^i)^2 + (\bar{y}_t^j - \bar{y}_t^i)^2$. The Jacobian of this function is given by

$$H_{TO_t} = \frac{\delta h_{TO}(\bar{T}_t^i, \bar{O}_t^j)}{\delta \bar{X}_t} \tag{14}$$

Each traffic user maintains the state of all objects it has within its state vector X. All relations between traffic users and moving objects are independent of each other, this results in the Jacobian matrix of function h that is given below.

$$H^i_{TO_t} = \begin{bmatrix} 0 & \dots & 0 & H^i_{RO_t} & 0 & \dots & 0 & H^j_{RO_t} & 0 & \dots & 0 \end{bmatrix} \tag{15}$$

Where

$$H^i_{TO_t} = \begin{bmatrix} \frac{-(\bar{x}^j_t - \bar{x}^i_t)}{\sqrt{q}} & \frac{-(\bar{y}^j_t - \bar{y}^i_t)}{\sqrt{q}} & 0 \\ \frac{-(\bar{y}^j_t - \bar{y}^i_t)}{\sqrt{q}} & \frac{-(\bar{x}^j_t - \bar{x}^i_t)}{\sqrt{q}} & -1 \end{bmatrix} \tag{16}$$

and

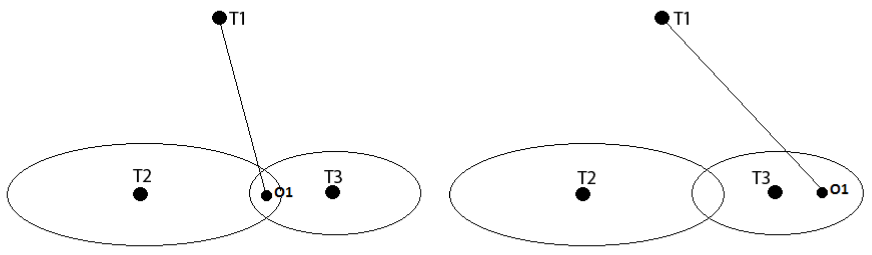
$$H^j_{TO_t} = \begin{bmatrix} \frac{-(\bar{x}^j_t - \bar{x}^i_t)}{\sqrt{q}} & \frac{-(\bar{y}^j_t - \bar{y}^i_t)}{\sqrt{q}} & 0 & 0 \\ \frac{-(\bar{y}^j_t - \bar{y}^i_t)}{\sqrt{q}} & \frac{-(\bar{x}^j_t - \bar{x}^i_t)}{\sqrt{q}} & 0 & 0 \end{bmatrix} \tag{17}$$

When the detected object $\bar{O}^j_t$ is another traffic user the Jacobian matrix will be of the same form as it is for $H^i_{TO_t}$. The matrix R^i_O is initialized to:

$$R^i_O = \begin{bmatrix} 1000 & 0 \\ 0 & 3 \end{bmatrix}$$

4.4 Multiple Hypotheses Tracking

All traffic users send their data to other traffic users by using the communication protocol described in section 3.3. Each traffic user generates a map containing the locations of all moving objects it has detected or received from other traffic users. The Multiple Hypotheses Tracking (MHT) algorithm as explained in [6] uses the previously explained Kalman filter to update the states of traffic users and moving objects.



(a) Two Hypotheses (b) One Hypothesis

Figure 16: Multiple Hypotheses Tracking

Suppose traffic user 1 (T1) detects a moving object O1, and is already tracking two other traffic users T2 and T3. In the left drawing of figure 16, O1 finds himself in both the uncertainty bounds of T2 and T3, therefore three data association hypotheses are generated: one in which O1 is used to update the state of traffic user T2, one in which O1 is used to update the state of traffic user T3, and one in which O1 represents a new moving object. These updates result in a branching of the Kalman filter into three different Kalman filters each of which represents a hypothesis. All hypotheses have a probability of being the right hypothesis and the MHT algorithm then chooses the most likely one. In the right drawing of figure 16, O1 is detected within the uncertainty bounds of T3 and the probability of O1 belonging to T2 is almost zero. Therefore, only two hypothesis are generated: O1 can either represent T3 or can represent a new moving object. The uncertainty bounds of both traffic users are given by the covariance matrix Σ. The probabilities of a hypothesis Ω are computed following [40], the formula is given below:

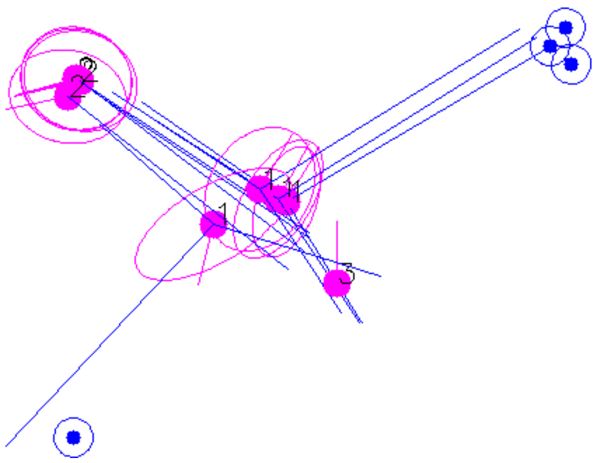
$$p(\Omega_t^k|Z_{1:t}) = p(\Omega_{t-1}, \psi^k|Z_{1:t}) \tag{18}$$

$$= \frac{1}{n} p(Z_t|\Omega_{t-1}, \psi^k) p(\psi^k|\Omega_{t-1}) p(\Omega_{t-1}|Z_{1:t-1}) \tag{19}$$

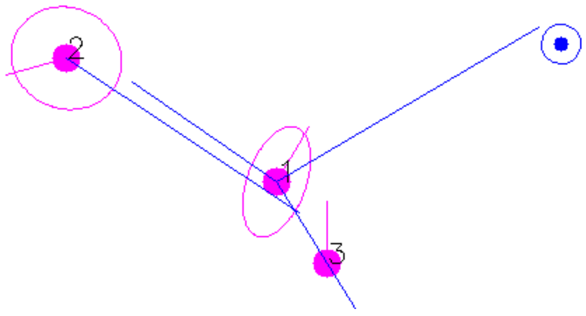
The k_{th} data association hypothesis at the current time step is denoted by ψ^k and n denotes a normalization constant. The probability of a hypothesis Ω depends on the measurements $Z_{1:t}$ obtained during the time between the first time step to time t.

4.4.1 Experiment

In the figure below two drawings are shown of a situation in which the multiple hypotheses algorithm is applied. During this experiment the robots explained in chapter 3 are used to represent traffic users. The drawings below are generated using the data obtained by robot 3. In drawing 17(a) robot 3 has multiple hypotheses about robot 1 and robot 2. Robot 1 thereby detected a moving object that is shown as a blue dot. The ellipses around the robots and the moving object show the uncertainty bounds. Robot 3 made four different hypotheses regarding the positions and rotations of both robot 1 and robot 2. The position of the moving object depends on the position and rotation of robot 1. Robot 3 therefore has four different possibilities of how the moving object can be placed.



(a) Multiple Hypotheses



(b) Resulting Hypothesis

Figure 17: Multiple Hypotheses Tracking: The pink dots represent the robots, the blue dots correspond to moving objects. The pink ellipses and blue circles represent the uncertainty bounds. The blue lines represent observations made by robots and the pink line initiating from a robot shows its rotation.

Depending on the previous positions and rotations and the current possibilities, the most likely hypothesis is chosen. The resulting hypothesis is shown in drawing 17(b).

Chapter 5

5 Simulator

The previous chapter explained how traffic users and moving objects are detected and tracked. While the results of the tracking system can be used as an input for the collision risk assessment algorithm, the algorithm is trained and tested using a data set created by a simulator. A simulator can create a large data set within a reasonable amount of time and performs automatic labeling of the data. By using a physics engine it is attempted to obtain data as realistic as possible. The first section describes the framework that is used for implementing the simulator. Section 5.2 explains the environment used in the simulator, next the implementation of the code is shortly described in section 5.3. Section 5.4 describes the physics used in the simulator, and section 5.5 elaborates on how path planning was done during simulation.

5.1 The Framework

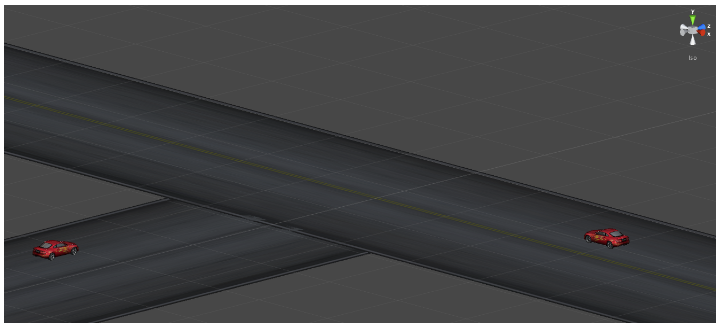
To implement the simulator the freely available game development software and game engine Unity is used. Unity provides a framework in both java and c# programming languages. Unity uses the OpenGL graphics library under Mac OSX and is able to work with DirectX under Windows. Unity also provides the possibility of importing 3D models that were designed using 3D modeling software such as Maya 3D. This alleviates the user from programming models in OpenGL or by having to map textures on objects generated in OpenGL. Unity therefore makes it easier for the user to work with more complex models. The framework also provides a physics engine with which physics can be simulated.

5.2 Environment

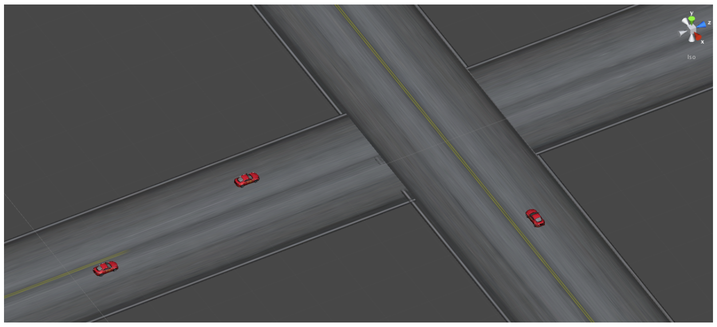
The simulation environment consists of roads and cars. The roads consist of planes that are drawn in Unity and a texture is used to give them the appearance of a road. A road has the same material property that asphalt has assigned to it. This material property defines the friction that a road has with objects that are situated on it. During simulation only two cars are used since only the interaction between two cars will be classified by the classifier. When more cars are apparent the classifier will classify the situation between each car separately and base its conclusion on the different outcomes between the different cars. Only one type of car is used in the simulator, this car is readily available from the Unity tutorials website [49] and can be seen in figure 18.

Figure 18: Two cars in the simulator

Within Unity the environment can easily be changed. Different road configurations are possible such as straight roads, normal intersections and t-intersections. In figure 19 two different road configurations can be seen.



(a) T-intersection



(b) Crossroads

Figure 19: Different road configurations

5.3 Implementation

The traffic simulator consists of three different parts: the code that simulates the car, the data retrieval code and the code that steers the car. The code that simulates the car sets up the physics of the car. The physics include the weight, the suspension, the maximum speed and acceleration of the car. The code also takes care of shifting gears and other necessary elements to enable the car to behave like a normal car. This part of the code also takes care of the updates that need to be done. Unity offers two different update methods. One of which is called *update* and the other one is called *fixedupdate*. The function *update* is called when the graphics card finished rendering a frame and is therefore called at variable time intervals dependent on the frame rate. The function *update* is mainly used to call to other functions that update the graphics of the game. The functions that update the physics of the car are normally called from within the function *fixedupdate*. This update function is independent of the frame rate and makes sure that the physics are updated at a constant time-interval of 0.02 seconds.

The second part consists of the data retrieval code. Information about the time, the position, the rotation and the status of a car are stored. The status of a car indicates if a car has crashed or not. This data is made relative to other cars and can then be used for training and testing of the collision risk assessment algorithm. More information about preprocessing of the data and the collision risk assessment algorithm is given in the next chapter.

The third part is the code that steers the car. A car can be manually controlled or it can be controlled by the computer. When a car is manually controlled the steering and throttle applied to a car can be recorded at each time step. The code can then automatically recreate the path when following the recorded input parameters, by adding noise to the recorded input unique paths can be created automatically. This part of the code is mainly used to build a data set for the collision risk assessment system, which is explained in the next chapter.

5.4 Physics

The Unity game engine has a built in physics engine that can be used to simulate gravity but also car physics and more. Car physics are used in this simulator to make the car behave more realistically. Cars used in the simulator are all assigned a rigid body and suspension. The rigid body is used to simulate mass and drag while the suspension is used to define the grip that the car has on the road. Wheels are thereby assigned wheel friction curves that define how much a tire is slipping both sideways and in the rolling direction. A collision plane is defined around each car and aside of the road. These planes make sure that cars can not drive through each other or through the wall. Whenever a collision occurs a signal is given to the script which is driving the cars. Figure 20 shows a collider on the side of a road. Colliders are invisible planes with which cars can collide and prevent the cars from driving of the road. In the figure below two colliders are made visible by a green wire-frame and by a light-grey surface.

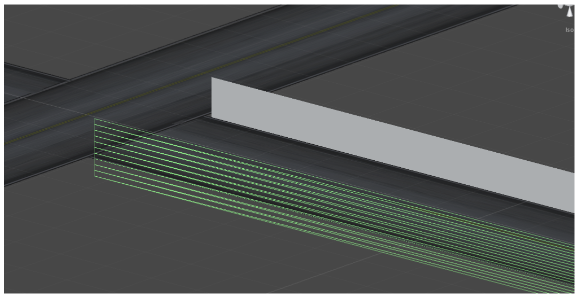


Figure 20: A roadside collider

5.5 Path Planning

In order to obtain a large data set, data generation should be done automatically. A data point consists of the path that a car has driven storing its rotation, speed, and position at every time step with time steps of 0.02 seconds. At the beginning of a session (the creation of one data point) each car selects a different starting position randomly. When a car has selected a starting position it notifies the other cars of its decision so the other cars will only choose a starting position that has not yet been taken.

The starting positions themselves are defined manually and must be recorded before a data set can be created. Tracks initiating from each starting position must also be defined manually. Each starting position generally has multiple tracks along which cars can drive. After a car has selected a starting position it randomly select a track that initiates from this starting position. Only a finite number of tracks can be created manually. To account for this limitation each car uses Unity's random number generator to adjust the path it is driving along. By slightly changing the previously recorded input, cars drive differently every session. A new session is started automatically when one of the cars crashes into the wall, when both cars reach the end of their track (safe situation) or when both cars crash into each other (dangerous situation).

5.5.1 Plane Detection

Plane detection is part of path planning and tries to avoid cars from crashing into the colliders on the side of the road. Data obtained during a session in which a car drives into a wall is not used to train the collision detection algorithm with, since we are mostly interested in interaction between other moving objects (traffic users) and not static objects. When a plane is detected the car slows down and tries to steer away from the plane. When the car is driving at high speed it might not have the chance to change direction anymore and will then still crash into the wall. The plane detection first determines the distance from the car to the plane and at which side of the plane the car is located. This distance is computed using the formula given in equation 20

$$d = \frac{(x_2 - x_1)(z_1 - z_0) - (x_1 - x_0)(z_2 - z_1)}{\sqrt{(x_2 - x_1)^2 + (z_2 - z_1)^2}} \tag{20}$$

The variables x_0 and z_0 are the x and z positions of the car (the y-axis represents height in Unity and is not used). The variables x_1 and z_1 and x_2 and z_2 are the minimum and maximum x and z coordinates of a collision plane. When the distance gets below a certain threshold the following formula determines if the car is driving toward or away from the plane.

$$ic = \frac{cx_p - cx_c + \tan(\alpha_c - \alpha_p \cdot cz_p)}{\tan(\alpha_c - \alpha_p)} \tag{21}$$

where

$$cx_p = x_p \cdot \sqrt{\cos(\alpha_p)^2} + z_p \cdot \sqrt{\sin(\alpha_p)^2} \tag{22}$$

$$cx_c = x_c \cdot \sqrt{\cos(\alpha_p)^2} + z_c \cdot \sqrt{\sin(\alpha_p)^2} \tag{23}$$

and

$$cz_c = x_c \cdot \sqrt{\sin(\alpha_p)^2} + z_c \cdot \sqrt{\cos(\alpha_p)^2} \tag{24}$$

In the above equations x_c represents the x coordinate of the car and x_p the x coordinate of the collision plane. The variable z_c and z_p represent the z coordinate of the car and the plane respectively. The variables α_c and α_p are the rotations of the car and the plane respectively. The variable cx_c represents the combined x coordinate of the car incorporating the x and z coordinate of the car with respect to the rotation of the car. The variable cx_p is the combined x coordinate of the plane incorporating the x and z coordinate and the rotation of the plane. The cz_c variable is the combined z variable for the car incorporating the x and z coordinate of the car with respect to the rotation of the car. The result of formula 21 (the variable ic) is the intersection coordinate which specifies at what coordinate the car will hit the plane when the car would continue to drive along the same line. Therefore, the results of formula 20 and 21 determine in which direction the car should steer to avoid hitting the plane.

Chapter 6

6 Collision Risk Assessment

Safety systems can be divided into two different categories: Passive safety systems and active safety systems. Passive safety systems aim to keep damage as low as possible in the case of an accident but have no means of preventing the accident (e.g. seat-belts and airbags). Active safety systems try to prevent accidents before they occur [20]. The system described in this thesis implements an active safety system which notifies traffic users when they find themselves in a dangerous situation. This chapter describes how collision risk assessment is performed to determine if a warning should appear to notify a traffic user of a dangerous situation. To make a well founded judgement of the current situation having reliable and complete information is a big advantage, complete information is however almost never available. By using communication between traffic users more complete information can be obtained and the range at which moving objects are detected can be enlarged. However, even by using communication no complete information can be obtained due to noise in sensor data and failures in the network that might occur from time to time.

This chapter is organized as follows. Section 6.1 explains the two different classes used for classification. Section 6.2 elaborates on the features that are used for classification. Section 6.3 explains what experiments have been conducted to obtain data for training and testing. The algorithm is explained in section 6.4 while in the last section the results of the experiments are given.

6.1 Classes

A warning needs to be given when a dangerous situation occurs. This results in two different classes: a class in which no warning should be given is named *safe situation* and a class in which a warning should be given is named *dangerous situation*. The class *dangerous situation* can however be broken down into different classes appropriate to the situation in which the traffic user finds himself at a certain moment. The advantage of breaking down the different dangerous situations is that it can give the user a specific warning about how he or she should react to the upcoming danger. Also, the state of the user at the time the warning is given can be taken into account. If the user is not paying attention a warning can be given. However, when warnings are given too often the user might start to ignore warnings. This report only discusses the capability of the collision avoidance system to classify the two classes *safe situation* and *dangerous situation*. The class *dangerous situation* is denoted by C_1 and the class *safe situation* is denoted by C_2. Class C_1 bears the label *1* and class C_2 the label *-1*.

6.2 Features

Collision patterns can be learned when using the right features. A few different features have already been proposed such as the vehicle maneuver, direction and angle in [43] and velocity vectors in [25]. This section explains the different features that will be used for classification. Features that are relative are always relative between two objects only. One of these objects being the object who is classifying the current situation (host) while the other object represents the object being detected by the host. In the current status of this project, objects only represent cars but can later also represent pedestrians and other traffic users. For each detected object the host runs the classification algorithm using the features described below as input. These features can be obtained using the sensors with which the car is equipped as described in chapter 3 and by using the communication algorithm.

6.2.1 Relative Distance

The distance between two objects is relevant when detecting dangerous situations. When two objects are closer to each other while in motion a dangerous situation might occur. When two objects are far away from each other the situation will be less dangerous. In order to determine the distance between two objects the relative position is required. The relative distance is a temporal variable and needs to be stored at each time step. The relative distance is given in millimeters.

6.2.2 Relative Speed

The relative speed between two objects determines at what rate two objects are moving closer to each other. When the relative speed between to objects is high and the distance is getting smaller this can be an indication for a possible dangerous situation. Relative speed is a feature that is often used such as in [25]. The relative speed changes over time and is therefore stored at every time step. The relative speed is given in millimeters per second.

6.2.3 Relative Angle

The relative angle between two objects gives information about how two objects are positioned with respect to each other. This is a relevant feature when combined with the relative direction and gives an indication under what angle the objects are approaching each other or under what angle they drive away from each other. The relative angle is given in degrees.

6.2.4 Relative Direction

The relative direction determines if two objects are approaching or leaving each other. This is determined by using the global angle of each object and by determining their current linear path. The point where both paths of both objects cross is called the intersection point. When the two objects are moving towards this point it is said that both objects are on a collision course, no matter the distance they are away from each other. When one of the objects is moving away from the intersection point it is unlikely for a collision to occur between the two objects. In figure 21 two black squares representing cars are visible. When both cars move towards the intersection point, a dangerous situation might occur. This feature will then take the value 1. When one of them moves away a dangerous situation is less likely and this feature will then take the value -1.

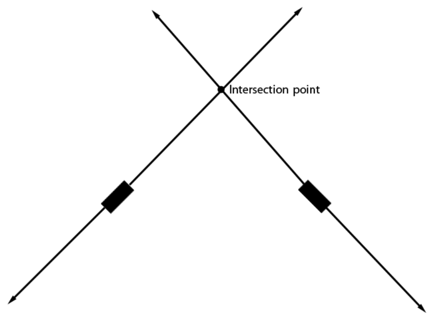


Figure 21: Relative direction

6.2.5 Relative Rotational Speed

When an object turns at a high rotational speed it is easier to loose grip and therefore get out of control. A dangerous situation might therefore occur. Since this feature is temporal it is stored at every time step. The relative rotational speed is given in degrees per second.

6.2.6 Next Time-Step Relative Distance

Since the speed of both of the objects is known the distance between the two objects at a later time step can be predicted. The size of this time step can easily be varied but is now taken 2 seconds. The next time-step relative distance is given in millimeters.

6.3 Data Retrieval

Collisions are difficult to simulate in the real world, it is therefore an acceptable method to obtain training data using computer simulations [43]. In this project, the simulator described in chapter 5 is used to obtain training and testing data for traffic accidents. The simulator only terminates when the user manually stops the simulation. During simulation two data files per session are generated, one for each car. Each session ends whenever an accident has happened, when both cars ran out of data points or when one of the cars hit the side of the road. A new session is then initiated automatically as described in section 5.5. Each data file consists of multiple lines. Per time-step, the class of the situation at that time step, the x, y and z coordinates of the car, the rotation of the car measured with respect to the y-axis and the current simulator time are stored. A time step is 0.02 seconds in size.

The class of the situation is safe (C_2) until an accident happens or until one of the cars hits the side of the road. When the situation stays safe throughout a session both cars classify the session as belonging to the class C_2. When an accident happened, both cars classify the session as belonging to class C_1 (unsafe). When one of the cars has hit the side of the road it communicates this to the other car, both cars then *label* the session as 2. This label does not belong to any class and only indicates that these sessions need to be discarded later on. Data files belonging to the same session always contain the same label.

The simulator time consists of a timer that is increased with 0.02 seconds every time step and is set to zero at the beginning of the simulation. In figure 22 the coordinate system of the simulator is displayed. The y-axis is perpendicular to the x- and z-axis and is not drawn in the figure. The degrees in the figure indicate the angle of rotation around the y-axis.

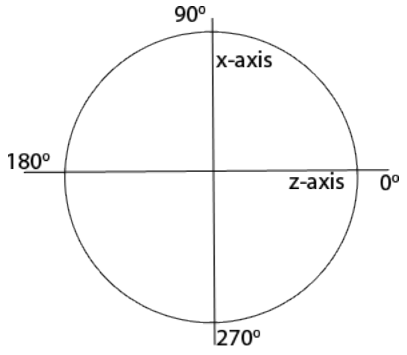


Figure 22: The coordinate system of the simulator

6.3.1 Feature Calculation

The data files generated by the simulator need to be preprocessed in order to obtain the features as described in section 6.2. Before features are calculated the *labels* of the data files are checked. When the label is 1 or -1 the data file belongs to class C_1 or C_2 respectively. When the label is 2 (i.e. one of the cars hit the wall) the data file does not belong to any class and will be discarded. During simulation 11281 sessions were executed, therefore the same amount of data files per car were recorded. Of these files, 1324 files were classified as a session in which an unsafe situation occurs. In 8347 of the files no dangerous situations occurs and are classified as safe. The other 1610 files were discarded. Features are calculated using a program written in c++. This section will describe per feature how it is calculated.

Relative Distance
To compute the relative distance between two cars the x- and z-position of both cars are used. Using formula 25 the euclidian distance between the two cars is calculated.

$$dist = \sqrt{(x_{car0} - x_{car1})^2 + (z_{car0} - z_{car1})^2} \tag{25}$$

Relative Speed
By storing the translation over the x and z axis of each car at each time-step, the speed of both cars can be calculated. The translation in x and the translation in z is calculated separately for each car given two sequential time-steps $t-1$ and t. By dividing these translations by the size of the time-step the speed in both directions can be calculated. The relative speed is calculated by taking the absolute value of the difference in speed in both directions between the two cars.

Relative Angle
To compute the relative angle the absolute value of the difference between the angle of both cars is used. When the relative angle is bigger than 180 degrees the angle is subtracted from 360 ($360 - relativeangle$) to obtain the smallest angle between the two cars.

Relative Direction
The relative direction is computed by first calculating the intersection as given in figure 21 using the intersection point between two linear equations. The distance to the intersection is calculated for two consecutive time steps for both cars. When the distance to the intersection at the last time step has become smaller the car is driving towards the intersection point. When both cars drive towards the intersection point a 1 is written to the feature file while when one of the cars drives away from the intersection-point, -1 is written to the file.

Relative Rotational Speed
By storing the rotation at two consecutive time steps for both cars the relative rotational speed can be computed. First, the rotational speed for each of the cars is calculated separately. The absolute value of the difference between the rotational speed is then taken as the value for the feature.

Next Time-Step Relative Distance
Since the speed of each car in both x and z direction is known, the locations of the two cars at a certain amount ahead in time can be predicted. This is done by using the formulas $x_{pred} = vx * time$ and $z_{pred} = vz * time$ in which x_{pred} and z_{pred} are the predicted x and z locations of a car and vx and vz are the velocities in the x and z direction respectively. The variable $time$ is the amount of time ahead of the current time at which the prediction is made, this variable is set to two seconds. Once the predicted locations of both cars are known, the

same procedure to compute the feature *relative distance* is followed to compute the relative distance at this future time-step.

For every time step a line is written in the feature file containing the features explained above, every line begins with indicating the label of the class to which the feature file belongs to. When a session ends in a crash every line in the feature file starts with a 1, otherwise every line starts with a -1. This means that every point in the feature file is either classified as safe or unsafe no matter how much time before the accident (if any) the data point was recorded.

6.3.2 Data-Set Preparation

As a result of the feature calculation process, one feature file per session is produced containing all features explained above for each time-step and the class to which the feature file belongs to. The classifier described in the next section needs to predict if the current situation is either safe or dangerous. The classifier is trained using the features explained in the previous section. However, these feature files are not yet in the right format and are thereby of different length since some sessions take longer than other sessions. In some sessions the cars crash sooner than in other sessions while in other sessions cars do not crash at all.

The data recorded at the end of a session is the most informative. When an accident has happened the data recorded in the last few seconds before the accident happened is very descriptive. Only in the beginning of a session are the feature values belonging to the unsafe class expected to be similar to the ones belonging to the safe class. The accident is still far away in time and therefore none of the features will indicate that an accident will happen in the future. When getting closer to an accident the differences between feature values for safe situations and dangerous situations is however expected to grow and a better separation between the both classes can be obtained, this phenomenon is described in subsection 6.5.1.

It is interesting to learn the prediction capacity of the classifier at different times leading up to an accident. Better predictions are expected when using data closer in time to an accident than when using data further away in time. At the end of each feature file either an accident happened or no accident happened. The end of a feature file is therefore most descriptive and is called the *ending point*. From this ending point on different windows are created that follow each other up in time. A window is either 0.5, 1 or 2 seconds in size and the beginning of each window is placed 0.2 seconds after the beginning of the previous window. This means that windows overlap each other. Each window contains the data of each feature file that was recorded during the time the window spans.

As an example, when using windows with a span of two seconds: the first window spans the time from zero seconds to the ending point to two seconds away from the ending point while the second window spans a time from 0.2 seconds from the ending point to 2.2 seconds from the ending point. Here, the ending point is the point where either an accident or nothing happened. In figure 23 a drawing of the example above is given.

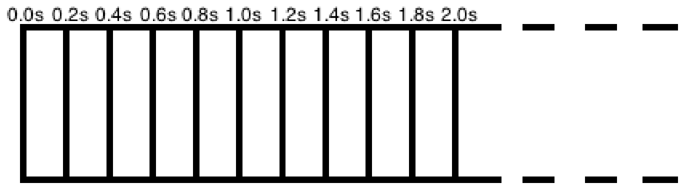


Figure 23: Data Windows

In figure 23, at zero seconds from the ending point the first window is initiated, this window ends at 2 seconds. The window contains data from all feature files since all feature files are long enough to contain information from the ending point to two seconds before the ending point. Each data point in a window consists of the data of one feature file that describes these two seconds. There are therefore as many data points in this window as there are feature files that can provide data for this window. The further away in time from the ending point the less feature files are long enough to provide data for the window belonging to that point in time, thus the less data points that window contains. The classifier is trained and tested per window. Part of the data in each window is used as a test set while the other part is used for training. Moreover this method is explained in the next section. An overview of the data preprocessing stage is given in figure 24.

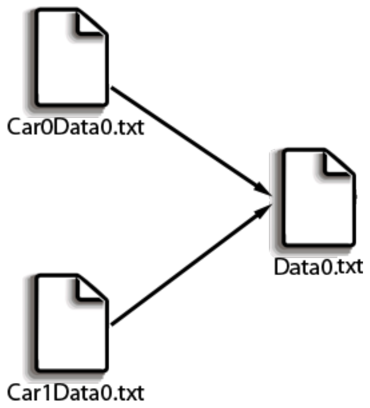


Figure 24: Data Preprocessing Pipeline

In the figure above the files Car0Data0.txt and Car1Data0.txt contain the data for the first session. During calculation of the features both files are merged into one file called Data0.txt (the feature file). Out of all data files and the therefrom generated feature files the windows are created.

6.4 Collision Risk Assessment Algorithm

During participation in traffic, data is obtained using the localization and tracking system as explained in chapter 4. By abstracting features from each data point and storing features for each data point for a certain amount of time steps the data can be classified. Different algorithms are applicable for classification of the data. Examples of such algorithms are hidden Markov models, parametric classifiers, and linear classifiers. However, hidden Markov models require a predefined model of which the transitions between states need to be learned. When using parametric methods the parameters of the distributions of the two classes need to be learned. A linear classifier does not require such data and only seeks to find the boundaries between two classes and is therefore a less complex approach. A Support Vector Machine (SVM) is therefore chosen to classify the data. This section first explains how the SVM classifier is implemented, it then elaborates on how the classifier is trained and tested.

6.4.1 Classification by an SVM

A support vector machine is a maximum margin method, this means that it tries to find the maximum margin when separating between two classes. The model is defined only by a subset of the training instances called the support vectors. The optimal separating hyperplane is the hyperplane that separates the two classes, defined in section 6.1, with the biggest possible margin. Class C_1 is defined as dangerous situation with label $r = +1$ while class C_2 is defined as safe situation bearing label $r = -1$.

The hyperplane is defined by a vector $\mathbf{w}$ orthogonal to the hyperplane and a threshold w_0. When using normal linear classifiers the following condition is sufficient for class 1 C_1: $g(x) > 0$ and for C_2: $g(x) < 0$. Therefore, instances lying on the positive side of the hyperplane belong to C_1 while instances on the negative side belong to C_2. The discriminant function $g(x)$ equals $\mathbf{w}^T\mathbf{x}^t + w_0$. In a support vector machine however the following conditions need to be met:

$$\mathbf{w}^T\mathbf{x}^t + w_0 \geq +1 \text{ for } C_1 \tag{26}$$

$$\mathbf{w}^T\mathbf{x}^t + w_0 \leq -1 \text{ for } C_2 \tag{27}$$

In the formula above $\mathbf{x}^t$ represents an instance out of N instances ($\{x^t\}_{t=1}^N$). The formulas above indicate that samples do not only have to be on the correct side of the hyperplane but should also be some distance away from the hyperplane. The larger the margin the better the generalization of the classifier. A figure explaining the margins and the support vectors is given in figure 25.

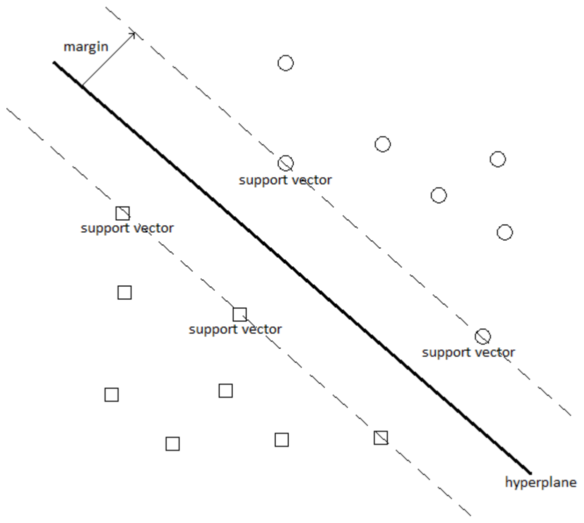


Figure 25: An optimal separating hyperplane

The distance of an instance $\mathbf{x}^t$ to the discriminant is

$$\frac{r^t(\mathbf{w}^T\mathbf{x}^t + w_0)}{||w||}$$

where $r \in \{-1, 1\}$, this function therefore always returns an absolute value if the instance is correctly classified. The distance should take at least the value ρ, the formula therefore becomes

$$\frac{r(\mathbf{w}^T\mathbf{x}^t + w_0)}{||w||} \geq \rho, \, \forall t$$

By setting $\rho||\mathbf{w}|| = 1$ and by minimizing $||\mathbf{w}||$ a unique solution can be obtained. From this follows that

$$min\frac{1}{2}||w||^2 \text{ subject to } r^t(\mathbf{w}^T\mathbf{x}^t + w_0) \geq +1, \, \forall t \tag{28}$$

By using the Lagrange multiplier λ_t equation 28 can be solved. The formula now becomes

$$\frac{1}{2}||\mathbf{w}||^2 - \Sigma_{t=1}^{N}\lambda_t[r^t(\mathbf{w}^T\mathbf{x}^t + w_0) - 1] \tag{29}$$

By minimizing the function with respect to $\mathbf{w}$ and $\mathbf{x}$ and by maximizing with respect to λ the solution can be found. SVM is implemented in Matlab using the matlab implementation of LIBSVM [10].

6.4.2 Kernel

When a linear classifier is not able to fit the data another hypothesis might be chosen such as a quadratic discriminant function. This however increases the space and time complexities. Kernels can be used to map data to a higher dimension where a linear model can actually fit the data [1]. It is assumed that the data is linearly separable given the high dimensionality of the data and does therefore not need to be mapped to another space.

6.4.3 Training of the SVM

The SVM is trained using the windows as explained in subsection 6.3.2. It is recommended that data is normalized before it is used for training and testing to avoid features in greater numeric ranges dominating the ones in smaller numeric ranges [10]. Each dimension is linearly scaled to fit within the range of [0, 1]. For training and testing 9672 feature files were used of which 4836 feature files are used for training and 4836 are used for testing. In the training set, 4167 instances belong to class C_2 and 669 belong to class C_1. The test set consists of 4181 instances belonging to C_2 and 655 instances belonging to C_1. A penalty factor c is determined which allows for a certain amount of the instances to lie within the margin of the SVM during training. This penalty factor can prevent the SVM from over-fitting the training data and therefore cause better generalization [1]. The parameter c is determined by using the following range: 2^{-4}, 2^{-2}, 2^{0}, 2^{2}, ... , 2^{16}. For each value in this range, five-fold cross validation is performed on the training set, the value of parameter c at which the best cross validation accuracy is achieved is chosen as the value for parameter c. The best accuracy is achieved when c is set to 4.

6.5 Results

This section displays the results of the classification algorithm. Previously the *ending point* was defined as the last point in a feature file which corresponds to the last point in time before either an accident happens or both cars ran out of data points while nothing happened. The results given in this section are all referring to this ending point as 0 seconds in time. It is expected that the further away in time from the ending point the less accurate the classifier becomes. Figure 26 shows this phenomenon. When making a prediction at the ending point an accuracy of 0.99 can be obtained. This accuracy however decreases when predictions need to be made further away in time from the ending point. The drop at five seconds occurs because cars normally crash within five seconds from the start of a new session, therefore, feature files containing accidents are at most five seconds long. At a distance greater than five seconds to the ending point only feature files that describe a safe situation are long enough.

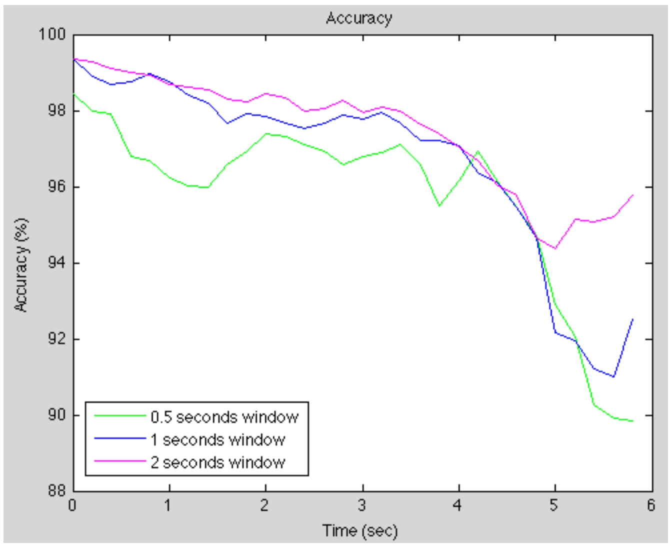


Figure 26: Accuracy on the testing dataset.

The accuracy in figure 26 however does not distinguish between the performance in the two different classes. It is important to look at the performance of the classifier for both different classes apart. There are far more instances belonging to the class safe situation (class -1). If the model obtained during training fits this class well, it is expected that the accuracy of the classifier is high even when the model doesn't fit the unsafe situation class (class 1) well. It is however important that the classifier performs well on both classes. When the classifier doesn't perform well on the safe situation class and classifies safe situations as dangerous, a driver might get annoyed by the system warning for unsafe situations while this is not the case. When the classifier doesn't perform well on the unsafe situation class and classifies unsafe situations as safe, a driver will get wrong information and the system does therefore not aid the driver in detecting dangerous situations. To evaluate the performance of both classes separately the true positive rate (tpr) and the true negative rate (tnr) are computed.

$$tpr = \frac{tp}{(tp + fn)} \tag{30}$$

$$tnr = \frac{tn}{(tn + fp)} \tag{31}$$

In formula 30 the equation to compute the true positive rate is given. In this formula tpr represents the true positive rate while tp is the amount of true positives during testing and fn represents the amount of false negatives during training. In formula 31 the equation to compute the true negative rate is given. Here tnr is the true negative rate and tn represents the amount of true negatives during training. The variable fp denotes the amount of false positives during training.

Figure 27 shows the true negative rate. It is expected that this rate stays constant since the feature values do not change much. In a safe situation the values stay about the same. It does however fluctuate and does not go up to 1.0 accuracy completely since some instances belonging to the *safe situation* class are still classified as belonging to the *unsafe situation* class. This can be explained by the fact that at some points in time the two cars are almost causing an accident and the recorded features values are thus very similar to the feature values that are recorded just before an accident. As explained before, for automation reasons a session is only classified as dangerous when the two cars hit each other in the end since this is undoubtedly a dangerous situation. However, when the two cars do not hit each other they can still get very close to each other and cause feature values that are similar to the ones belonging to the ones of an accident. These instances are therefore causing a false positive. This false positive is actually a true positive (a dangerous situation occurred) but is not classified as such since it belongs to a session in which the cars did not crash. This therefore causes the true negative rate to fluctuate.

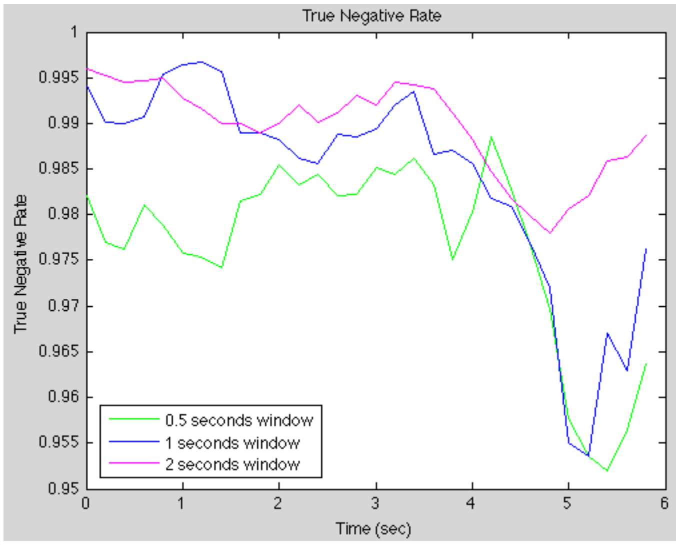


Figure 27: True Negative Rate

The true positive rate is given in figure 28. From this figure it can be seen that the true positive rate does decrease as the window used for classification is further away in time from the ending point. Just before the accident happens, dangerous situation are detected with a true positive rate of 0.98, while at 5 seconds before the accident the rate has decreased to 0.7.

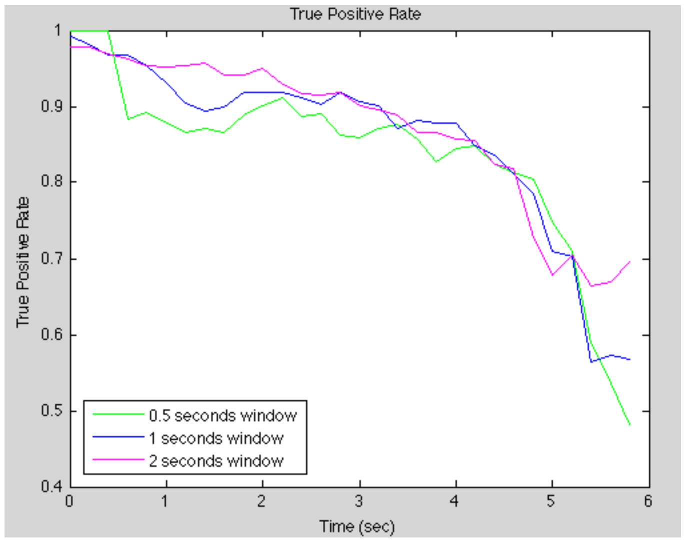


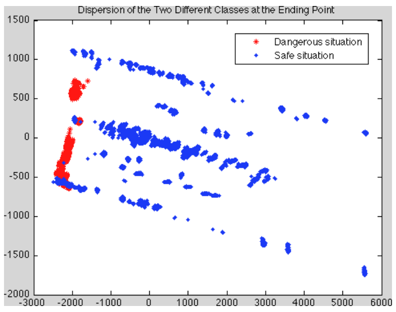
Figure 28: True Positive Rate

Figure 28 also shows fluctuations in the true positive rate. It is expected that the further away in time from the ending point the worse the predictive capacity of the classifier becomes. The fluctuations in accuracy are caused due to false negatives. As explained before, none of the sessions generated by the simulator are of the same length. In most cases, the ones in which no accidents happen are longer than the ones in which accidents happen. In some cases, when no accidents happen, cars can still get dangerously close to each other. This, however, does normally not happen at the end of a session but a few seconds before the end, this can cause the accuracy to decrease and then grow again.

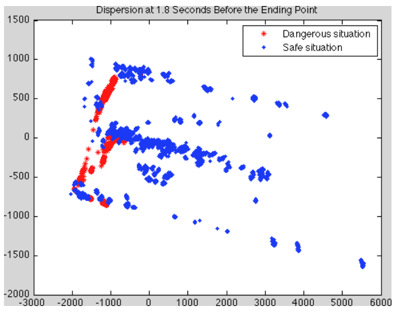
An example of such a session is one in which two cars approach the crossroad but still have a respectable distance between each other. Once both cars have entered the crossroad they pass each other with only few centimeters distance, after both cars leave the crossroad the distance between the two cars gets bigger again. Once both cars ran out of data points to follow (the *ending point*) the session will stop. The situation described in the example above happens mostly between 3 and 4 seconds towards the ending point. The windows that cover this part therefore contain instances that belong to the class *safe situation* that have values that are close to the values of instances belonging to the *unsafe situation* class. It is therefore likely that after training a model is obtained that classifies the instances belonging to the *unsafe situation* as belonging to the class *safe situation* causing the true positive rate to drop. After 4 seconds this phenomenon happens less often causing a different model and the true positive rate to increase again.

6.5.1 Class Dispersion

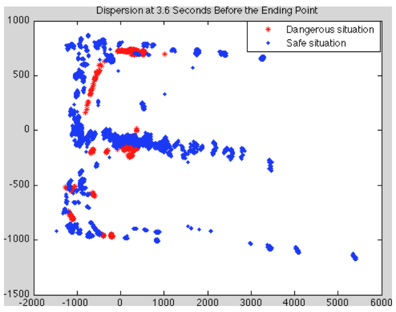
As explained before in subsection 6.3.2 it is expected that the two classes are less well separable using data further away in time from the ending point than when using data closer to the ending point. The figures below show the dispersion of this data at different times measured from the ending point.



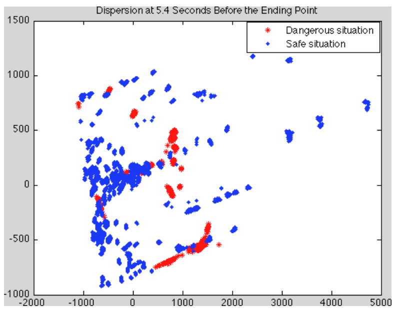
(a) Dispersion at 0 seconds to ending point



(b) Dispersion at 1.8 seconds



(c) Dispersion at 3.6 seconds



(d) Dispersion at 5.4 seconds

Figure 29: Dispersion of the two principle components with the highest variance at different times towards the ending point.

To show this phenomenon principal component analysis has been performed on the data in different windows of different time steps. The two features with the highest variance are displayed in figure 29 at four different points in time. The graphs do indeed show that the two classes become less well separable.

Chapter 7

7 Discussion & Future Work

The work described in this thesis presents a framework that implements a collision warning system. In the introduction the following research questions were asked:

1. Can a SVM predict upcoming dangerous situations?

2. If the answer on question 1 is positive: What is the predictive capacity of the classifier using the 2 second rule?

The results in section 6.5 show that an SVM can predict an upcoming accident and therefore a dangerous situation. The predictive power does, as expected, decrease as predictions are made further away in time from an accident. As explained in the introduction, the 2-seconds rule is used by governments to give an indication to the driver of the minimal distance that a car should keep to a car driving in front of it. Two cars should pass by the same static landmark with at least a two seconds time difference, this gives the drivers enough time to react on any sudden changes. At two seconds before the accident happens, the algorithm can predict the upcoming accident about 94 percent of the time (true-positive rate at 2 seconds is 0.94). Too many false alarms can lead to annoyance of the driver, a high true-negative rate is thus important. The true-negative rate is about 0.98 and stays rather constant. This result is expected because unlike a dangerous situation, a safe situation does not have to be predicted. Whenever there is no accident in the near-future, each time step is classified as safe.

The risk assessment algorithm is trained and tested by using data created by a simulator. The format of this data is similar to the data that can be obtained in the real-world. It is likely that the algorithm can perform with about the same accuracy in the real-world as it does when using simulator data given that the data obtained by the sensors is noise free. This is however never the case and the performance of the algorithm is expected to decrease as the noise in the sensors increases. Different methods exist that can be used to obtain real-world training and testing data. One method is explained in chapter 4 of this thesis, however, image sequences can also be used to obtain information about dangerous situations and safe situations. These image sequences can be extracted from dashboard cameras or traffic surveillance cameras. By using image processing techniques the required features can be obtained from the image sequences. Another method is the use of crash reports created by the police. These reports describe accidents in detail and the features can possibly be extracted from a number of these reports. Both the use of images or crash reports are time consuming methods to implement and it was considered more important to assess the performance of the collision risk assessment algorithm using simulated data before implementing such data retrieval methods.

7.1 Human Factors

Human factors are important to bear in mind when developing a driver warning system. This thesis only focused on the implementation of such a system and one of the next steps should be the consideration of human factors. A safety warning system should be of aid to the driver and should not distract the driver while driving. False alarms caused by faulty sensor measurements can cause drivers to have an aversion to the safety warning system.

Ben-Yaacov et. al. state in [7] that there are two human factor issues that need to be kept in mind when developing a safety warning system. The first issue consists of the interface between the safety warning system and the driver, the second issue involves an analysis of the interaction between the driver and the safety warning system when both are capable of error. Their paper states that both a visual interface and an auditory interface are effective to some degree. The influence of a safety warning system was tested by evaluating the distance that drivers maintain to other drivers with and without using the warning system. They found that the use of a more reliable system will improve safety when the driver performance with the system increases. They also found that the safety warning system can help drivers to maintain proper distance to other cars.

Parasuraman et. al. state in [36] that poorly designed warning systems in aircraft encourages a pilot to not use the system due to a high false alarm rate and that this phenomenon is also apparent among drivers. They state that the need for an early detection need to be balanced with the avoidance of false alarms to prevent nuisance by the warning systems.

In [45], Seiler et. al. state that a collision warning system should be accepted by the driver. They state that an increase in warning frequency might desensitize the driver and that the driver might ignore future warnings. Thereby, sudden warnings can cause the driver to get distracted by the warning system during critical situations. Instead of giving out random warnings, a gradually changing interface should be used to get the driver acquainted with the warning system. Also, a safety warning system should adapt to the driver. A warning system fit for a passive driver will give many warnings to an aggressive driver which, as mentioned above, will desensitize the driver. A warning system fit for an aggressive driver might not alarm a passive driver in time.

7.2 Communication

Communication effectively aids the collision warning system by increasing the range at which moving objects can be detected. Moving objects that would normally have been invisible to a traffic user are now shared over a network. Communication can however also be used for malicious purposes. A traffic user can broadcast malicious data that indicates the locations of moving objects that do not exist. This will cause the safety warning system to give false alarms. When investigating a crash between two traffic users, a communication log can be used. However, a fake position of a traffic user can be broadcasted to affect the log on all traffic users involved in the accident, therefore manipulating the outcome of the investigation. More scenarios are possible in which breaking in on the communication network can be appealing. It is therefore likely that people are not willing to communicate data to others. It is thus important to increase the safety of such communication networks. Golle et. al. describe an approach in [18] to evaluate the validity of data broadcasted in a VANET. When receiving data, a node (such as a traffic user) tries to find an explanation for the received data keeping in mind that malicious nodes might be apparent. When one of the explanations of the data is consistent with the node's model of the VANET it will give this explanation a higher score. Yan et. al. describe a VANET security system in [51]. Their system uses a radar system to detect neighboring vehicles to confirm their broadcasted location. They furthermore assume that a majority of the traffic users is honest and broadcast their measured location.

7.3 Future Work

The framework described in this thesis is developed and tested at different stages: the communication part, the localization and tracking part and the collision risk assessment algorithm. At the moment of writing, the localization and tracking part, and the communication part are integrated. The collision risk assessment algorithm does however still rely on data obtained from the simulator. A next step towards the safety warning system is the integration of all parts into one system. Due to time limitation this was not possible during this project. When the system proves to be effective on the test robots it can be installed and tested on other transportation methods such as cars.

The effects of the system on human drivers should also be determined. After initial testing on the system on cars it should be evaluated how warnings should be given to the traffic users. The system can learn to adjust to the traffic user and give appropriate warnings depending on the vehicle and the person itself.

The safety warning system should become available for multiple types of vehicles, therefore, size and ease of installation is of concern. Not all traffic users can bear a laser scanner. Also, the present computing power should be packaged to fit small vehicles or even pedestrians. Other than the user interface, the system should be as invisible as possible. Therefore, as a last step the system should be packaged as small and as universal as possible.

References

[1] Alpaydin E., 2010. *Introduction to Machine Learning Second Edition*, The MIT Press.

[2] Agrawal M., Konolige K. 2006. *"Real-time Localization in Outdoor Environments using Stereo Vision and Inexpensive GPS."*, 18th International Coference on Pattern Recognition (ICPR), Vol. 3, pp. 1063-1068.

[3] Aköz Ö., Karsligil M. E. 2011. *"Traffic Event Classification at Intersections Based on the Severity of Abnormality."*, Machine Vision and Applications, pp. 1-20.

[4] Bachir A., Benslimane A. 2003. *A Multicast Protocol in Ad hoc Networks Inter-Vehicle Geocast.* The 57th IEEE Semiannual Vehicle Technical Conference, Vol. 4, pp. 2456-2460.

[5] Blackman S. 2004. *"Multiple Hypothesis Tracking for Multiple Target Tracking."* A & E Systems Magazine Vol. 19, No. 1, pp. 5-18.

[6] Blackman S., Popoli R. 1999. *"Design and Analysis of Modern Tracking Systems."*, Artech House, Boston.

[7] Ben-Yaacov A., Maltz M., Shinar D. 2002. *"Effects of an In-Vehicle Collision Avoidance Warning System on Short- and Long-Term Driving Performance"*, Human Factors, Vol. 44, No. 2, pp. 335-342.

[8] Boet M., dos Santos A. L. 2008. *"RFID Tags: Positioning Principles and Localization Techniques."*, 1st IFIP Wireless Days, pp.1-5.

[9] Borges G. A., Aldon M. J. 2004. *"Line Extraction in 2D Range Images for Mobile Robotics."*, Journal of Intelligent and Robotic Systems 40, pp. 267-297.

[10] Chang C. C., Lin C. J. 2011. *LIBSVM: A Library for Support Vector Machines.*, ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27.

[11] Dagan E., Mano O., Stein G. P., Shashua A. 2004. *"Forward Collision Warning with a Single Camera"*, Proceedings of IEEE Intelligent Vehicle Symposium, pp. 37-42.

[12] Chou T. W. 2012. *"Range Assisted HOG Pedestrian Detection"*, Department of Computer Science and Information Engineering, National Taiwan University.

[13] Dollar P., Wojek C., Schiele B., Perona P. 2012. *"Pedestrian Detection: An Evaluation of the State of the Art."*, IEEE Transactions on Pattern Analysis And Machine Intelligence, Vol 34, No. 4, pp. 743-761.

[14] Edgar P., Harris C. J. 1996. *"An Intelligent Driver Warning System for Vehicle Collision Avoidance"*, IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, Vol. 26, No.2 pp. 254-261.

[15] Fischler M. A., Bolles R. C. 1981. *"Random Sample Consensu: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography"*, Communications of the ACM, 24(6), pp. 381-395.

[16] Gangisetty R. 1997. *"Advanced Traffic Management System On I-476 in Pennsylvania"*. In Proc. IEEE ITSC '99, pp. 373-378.

[17] *"GPS.gov: Augmentation Systems."*, National Coordination Office for Space-Based Positioning, Navigation, and Timing. Retrieved May 9, 2012, from http://www.gps.gov/systems/augmentations/

[18] Golle P., Green D., Staddon J. 2004. *"Detecting and Correcting Malicious Data in VANETs"*, Proceedings of the 1st ACM international workshop on Vehiclar ad hoc Networks, doi:10.1145/1023875.1023881.

[19] Huang X., Janaswamy R., Ganz A. 2006. *"Scout: Outdoor Localizatoin Using Active RFID Technology."*, Proceedings of BROADNETS, pp. 1-10.

[20] Huang C. M., Lin S. Y., Yang C. C., Chou C. H. 2009. *"A Collision Pre-Warning Algorithm based on V2V Communication."*, Proceedings of the International Conference on Ubiquitous Information Technologies and Applications, Riccione, Italy, pp. 1-6.

[21] Huang C. M., Lin S. Y. 2011. *"An Early Collision Warning Algorithm for Vehicles Based on V2V Communication"*, International Journal of Communication Systems, Nr. 25, pp. 779-795.

[22] Jones W. D. 2001. *"Keeping Cars From Crashing."*, IEEE Spectrum, vol. 38, no 9, pp. 40-45.

[23] Maybeck P. S. 1979. *"Stochastic Models, Estimation and Control"*, Academic Press, Inc., New York, USA.

[24] Kalman R. E. 1960. "A New Approach to Linear Filtering and Prediction Problems", Journal of Basic Engineering 82 (1), pp. 35-45.

[25] Kamijo S., Matsushita Y., Katsushi I., Sakauchi M. 2000. *"Traffic Monitoring and Accident Detection at Intersections"*, IEEE Transactions on Intelligent Transportation Systems, Vol. 1, No. 2, pp. 108-118.

[26] Karthikumar P., Krishnaveni V. 2012. *"Survey of Routing Protocols in Vehicular Ad-Hoc Network"*, International Journal of Engineering Innovation & Research, Volume1, Issue 2, pp. 110-114.

[27] Li F., Wang Y. 2007. *"Routing in Vehicular Ad Hoc Networks: A Survey"*. IEEE Vehicular Technology Magazine, Vol 2, pp. 12-22.

[28] Lu F., Milios E. 1997. *"Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans"*, Journal of Intelligent and Robotic Systems 18, pp. 249-275.

[29] Maihöfer C. 2004. *"A Survey Of Geocast Routing Protocols"*. IEEE Communications Surveys & Tutorials, Vol. 6, pp. 32-42.

[30] Miller H. J., Shaw S. L. 2001. *"Geographic Information Systems for Transportation"*, Oxford University Press, New York.

[31] Miller R., Huang Q. 2002. *"An Adaptive Peer-to-Peer Collision Warning System"*, IEEE Vehicular Technology Conference (VTC), Vol. 1, pp. 317-321.

[32] Misener J. A., Sengupta R. 2005. *"Cooperative Collision Warning: Enabling Crash Avoidance with Wireless Technology"*, 12th World Congress on ITS, paper 1960.

[33] Mittag J., Thomas F., Härri J., Hartenstein H. (2009). *"Comparison of Single- and Multi-hop Beaconing VANETs"*. VANET '09 Proceedings of the sixth ACM international workshop on Vehicular Internetworking. pp. 69-78

[34] Montemerlo M., Becker J., Bhat S., Dahlkamp H., Dolgov D., Ettinger S., Haehnel D., Hilden T., Hoffmann G., Huhnke B., Johnston D., Klumpp S., Langer D., Levandowski A., Levinson J., Marcii J., Orenstein D., Paefgen J., Penny I., Petrovskaya A., Pflueger M., Stanek G., Stavens D., Vogt A., Thrun S. 2008. *"Junior: The Stanford Entry in the Urban Challenge"*, Journal of Field Robotics 25(9), pp. 569-597.

[35] Morioka Y., Sota T., Nakagawa M. 2000. *"An Anti-Car Collision System Using GPS and 5.8GHz Inter-Vehicle Communication at an Off-Sight Intersection"*, Proceedings IEEE Vehicle Technology Conference, vol. 5, pp. 2019-2024.

[36] Parasuraman R., Hancock P. A., Olofinboba O. 1997. *"Alarm Effectiveness in Driver-Centered Collision-Warning Systems"*, Ergonomics 40:3, pp. 390-399.

[37] Parker R., Valaee S. 2006. *"Vehicle Localization in Vehicular Networks."*, Vehicular Technology Conference, IEEE 64th, pp. 1-5.

[38] Peden M., Scurfield R., Sleet D., Mohan D., Hyder A. A., Jarawan E., Mathers C. 2004. *"World Report on Road Traffic Injury Prevention"*, World Health Organization, Geneva.

[39] Premebida C., Monteiro G., Nunes U., Peixoto P. 2007. *A Lidar and Vision-based Approach for Pedestrian and Vehicle Detection and Tracking"*, Proceedings of the 2007 IEEE Intelligent Transportation Systems Conference, pp. 1044-1049.

[40] Reid D. 1979. *"An Algorithm for Tracking Multiple Targets*, IEEE Transactions on Automatic Control, 24(6), pp. 843-854.

[41] Rusinkiewicz S., Levoy M. 2001. *"Efficient Variants of the ICP Algorithm"*, Proceedings of 3DIM 2001, pp. 145-152.

[42] Salim F. D., Loke S. W., Rakotonirainy A., Srinivasan B., Krishnaswamy S. 2007. *"Collision Pattern Modeling and Real-Time Collision Detection at Road Intersections"*. Proceedings of the 2007 IEEE Intelligent Transportation Systems Conference, pp. 161-166.

[43] Salim F. D., Loke S. W., Rakotonirainy A., Krishnaswamy S. 2007. *"U&I Aware: A Framework Using Data Mining and Collision Detection to Increase Awareness for Intersection Users"*, 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07), Vol. 2, pp. 530-535.

[44] Schagrin M., Fehr W. Retrieved December 17, 2012. *"Connected Vehicles Dedicated Short Range Communications Frequently Asked Questions"*, RITA, http://www.its.dot.gov/DSRC/dsrc_faq.htm.

[45] Seiler P., Song B., Hedrick J. K. 1998. *"Development of a Collision Avoidance System"*, University of California-Berkeley.

[46] Bar-Shalom Y., Xiao-Rong L. 1995. *"Multitarget-Multisensor Tracking: Principles and Techniques"*, 3rd printing, Urbana, IL: YBS.

[47] Thrun S., Burgard W., Fox D. 2005. *"Probabilistic Robotics"*, The MIT Press Cambridge, Massachusetts.

[48] *“The Two-Second Rule”*, Retrieved December 3, 2012, Road Safety Authority Ireland, http://www.rotr.ie/rules-for-driving/speed-limits/speed-limits_2-second-rule.html.

[49] Echterhoff J. *“Car Tutorial”*, Unity, Retrieved September 20, 2012., http://unity3d.com/support/resources/tutorials/car-tutorial

[50] Wang C. C., Thorpe C. 2002. *“Simultaneous Localization and Mapping with Detection and Tracking of Moving Objects”*, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Vol. 3, pp. 2918-2924.

[51] Yan G., Olariu S., Weigle M. C. 2008. *“Providing VANET Security Through Active Position Detection”*, Computer Communication, Vol. 31, pp. 2883-2897.

[52] Yang S. W., Wang C. C. 2011. *“Simultaneous Egomotion Estimation, Segmentation, and Moving Object Detection.”*, Journal of Field Robotics 28(4), pp. 565-588.

[53] Ye F., Adams M., Summit R. 2008. *“V2V Wireless Communication Protocol for Rear-End Collision Avoidance on Highways”*, Proceedings vehi-mobi workshop, IEEE ICC, pp. 375-379.

[54] Zou Y., Shi G., Zhao H. 2011. *“Traffic Incident Classification at Intersections Based on Image Sequences by HMM/SVM Classifiers”*, Multimedia Tools Applications, Nr. 52, pp. 133-145.