

# Opvangen en analyseren van data omtrent passagiersaantallen op treinen

Laurent Loots

Promotoren: prof. dr. ir. Filip De Turck, dr. Bruno Volckaert

Begeleiders: dr. Bruno Volckaert, dr. ir. Stefan Bouckaert, ir. Thomas Dupont, ir. Wannes Kerckhove

Masterproef ingediend tot het behalen van de academische graad van  
Master of Science in de industriële wetenschappen: elektronica-ICT

Vakgroep Informatietechnologie  
Voorzitter: prof. dr. ir. Bart Dhoedt  
Faculteit Ingenieurswetenschappen en Architectuur  
Academiejaar 2016-2017





# Opvangen en analyseren van data omtrent passagiersaantallen op treinen

Laurent Loots

Promotoren: prof. dr. ir. Filip De Turck, dr. Bruno Volckaert

Begeleiders: dr. Bruno Volckaert, dr. ir. Stefan Bouckaert, ir. Thomas Dupont, ir. Wannes Kerckhove

Masterproef ingediend tot het behalen van de academische graad van  
Master of Science in de industriële wetenschappen: elektronica-ICT

Vakgroep Informatietechnologie  
Voorzitter: prof. dr. ir. Bart Dhoedt  
Faculteit Ingenieurswetenschappen en Architectuur  
Academiejaar 2016-2017



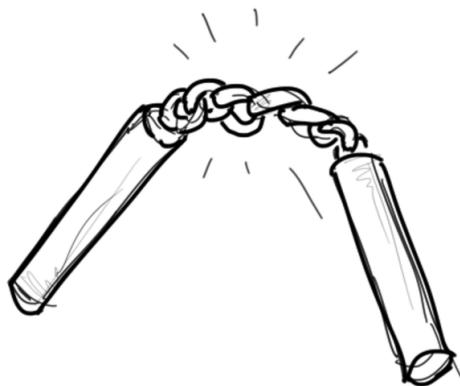
# Woord vooraf

Dit project is een geweldige uitdaging geweest om het laatste jaar van mijn ingenieursstudies mee af te sluiten. Het heeft me veel bijgebracht over beeldverwerkings- en netwerktechnieken, meer dan ik aan de start van het project had geanticipeerd, en heeft mijn vaardigheid met C++ en Linux omgevingen een enorme boost gegeven. Daarom had ik graag Televic Rail en IBCN bedankt om deze thesis aan mij toe te wijzen en deze leerrijke en boeiende ervaring te bezorgen.

Vanzelfsprekend neem ik ook graag de tijd om mijn begeleiders, dr. ir. Stefan Bouckaert, ir. Thomas Dupont, ir. Wannes Kerckhove en in het bijzonder dr. Bruno Volckaert te bedanken voor hun continue feedback en ondersteuning doorheen heel het project en het aanleveren van nodige hardware om de software mee te testen. Hun inbreng heeft me enorm geholpen om dit project tot een goed einde te brengen.

Tot slot bedank ik mijn ouders, Alain en Christine, voor hun steun en begrip tijdens de vele moeilijke programmeersessies in het begin van het project. Zij hebben enorm geholpen om mijn gedachten even te verzetten op frustrerende momenten wanneer de zoveelste onverwachte geheugenlek of semgentatiefout de kop op stak. In het bijzonder wil ik mijn vader bedanken, die de moeite nam om de afgewerkte thesis nog eens van begin tot einde door te lezen.

Laurent Loots, 2 juni 2017



C++ is a set of nunchucks, powerful and impressive when wielded but takes many years of pain to master and often you probably wish you were using something else.

# Toelating tot bruikleen

"De auteur geeft de toelating deze masterproef voor consultatie beschikbaar te stellen en delen van de masterproef te kopiëren voor persoonlijk gebruik.

Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze masterproef."

Laurent Loots, 2 juni 2017

# **Opvangen en analyseren van data omtrent passagiersaantallen op treinen**

door  
Laurent LOOTS

Masterproef ingediend tot het behalen van de academische graad van Master of Science in de  
industriële wetenschappen: elektronica-ICT

Academiejaar 2016 - 2017

Promotoren: prof. dr. ir. Filip De Turck, dr. Bruno Volckaert  
Begeleiders: dr. Bruno Volckaert, dr. ir. Stefan Bouckaert, ir. Thomas Dupont, ir. Wannes  
Kerckhove

Faculteit Ingenieurswetenschappen en Architectuur

Vakgroep Informatietechnologie, voorzitter: prof. dr. ir. Bart Dhoedt

## **Abstract**

De laatste tijd is er een stijgende vraag naar automatische passagierstelsystemen voor treinen. Dergelijke systemen voeren automatische tellingen van passagiers uit op halteniveau, wat een treinoperator inzicht kan geven in de bezetting en populariteit van bepaalde trajecten en haltes. Met deze informatie kunnen zij hun dienstregeling optimaliseren.

In deze thesis wordt een volledig *automated passenger counting* systeem ontwikkeld op basis van dieptecamera's. Dit systeem bestaat uit drie componenten: software voor het aansturen van een dieptesensor met behulp van een microcomputer, software die de microcomputers kan aansturen en op bestaande infrastructuur van de trein kan worden uitgevoerd en een back-end om de tellingsresultaten te visualiseren op een betekenisvolle manier. Bij ontwikkeling van het systeem zal worden gezocht welke bestaande softwarepakketten er kunnen worden gebruikt om deelproblemen aan te pakken zoals netwerkdiscovery, videotransfer over het netwerk en communicatie met bestaande boordsystemen op de trein.

Het systeem dat ontwikkeld werd komt tegemoet aan de verschillende uitdagingen van een trein-infrastructuur zoals het mogelijk wegvalen van netwerkverbindingen, dynamische samenstelling van treinstellen en beperkte bandbreedte naar de buitenwereld. Daarnaast zal de performantie, schaalbaarheid en verbruikte systeembronnen (processorkracht, netwerkverbruik en geheugenverbruik) van de verschillende componenten worden opgemeten waarvan toepassing. Tot slot zullen enkele voorstellen tot verbetering en uitbreiding in detail worden besproken.

**Trefwoorden:** Automated Passenger Counting, trein, implementatie, stereovisie

# Implementation of an Automated Passenger Counting system for trains

Laurent Loots

Supervisors: dr. Bruno Volckaert, dr. ir. Stefan Bouckaert, ir. Thomas Dupont, ir. Wannes Kerckhove

**Abstract**—This paper describes the development and evaluation of an automated passenger counting (APC) system for trains using stereographic cameras. First, the scope and challenges of the APC system will be discussed. Then the implementation of the different software components will be discussed in detail. Afterwards the different software components will be evaluated in terms of performance and scalability. Finally, some ideas for further research will be listed.

**Index Terms**—APC, automated passenger counting, trains, implementation, stereovision

## I. INTRODUCTION

ATELY there is a growing demand for automated passenger counting systems on the railway market. Such systems can give train operators valuable insights in the popularity of their trips. APC – systems are popular because they can be very accurate and provide insights on a station-per-station level.

To develop an APC – system, three components are needed: a sensor that detects passengers boarding and getting off and communicates the data via a local network, central server on the train (traininside server) that gathers the sensor data, applies a passenger detection algorithm and sends the counting results to an external server outside the train (wayside server) and a back-end to visualize the counting results in a meaningful way.

In this implementation, a stereographic camera will be used as sensor, which will be connected to an ARM – based microcomputer to communicate depth images over the network. This microcomputer will run a piece of software named camNode. The software that will gather the depth images from the microcomputers, named camServer, will be run on the existing infrastructure. Both camNode and camServer will be developed in C++ for performance reasons and will be run in a Linux environment. Finally, the back-end will be written in PHP so it'll be easily accessible to system

administrator and analysts.

While developing the APC system, several problems will need to be addressed [1]. First, there's the dynamically changing composition of trains, which will make a frequent discovery of APC hardware necessary. Next, there's the problem of limited bandwidth with the wayside server (sometimes none at all for short periods of time). Also, the system will need to be able to detect several errors (network and hardware related) and report them to the wayside server. Finally, there's the problem of limited computing resources on the existing train infrastructure where the camServer wil run on.

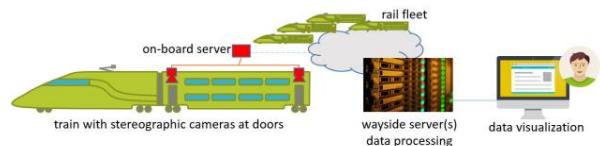


Figure 1: overview of APC components [1]

## II. DEVELOPMENT OF CAMNODE

The camNode software needs to achieve 6 things: detecting camera hardware connected to the microcomputer, generating depth images from 2 RGB cameras, sending depth images to a camServer instance, receiving commands from a camServer instance (for error reporting and gathering metadata), announce itself in the local network and manage some metadata in a configuration file.

### A. Hardware detection

In Linux, cameras are accessible through system paths (such as /dev/video0). Linux has a built-in API called V4L2 that allows the user to check if a certain video device is available on a system path. The solution proposed here is based on a tutorial by JWH Smith [7]. Using this library, we can iterate over the different paths (/dev/video0 to /dev/videoX, with X being a fixed upper limit) to see if a camera responds. If it does, it can be initialized and marked as ‘available’ to the rest of the software.

To get a videofeed from the camera, the VideoCapture class from the OpenCV [8] library will be used. This class is built on V4L2 and uses the same indices to address a video camera, but is not able to unambiguously tell if a camera is connected or not on a certain index.

### B. Generating depth images

Two images taken at the exact same time but from slightly different camera perspectives can be used to generate an estimated depth map of the scene. In order to achieve the depth map, the intrinsic-, extrinsic- and distortionparameters of the camera pair have to be known. These parameters can be calculated by calibrating the camera pair, using picture pairs of a calibration pattern such as a checkerboard with known size.

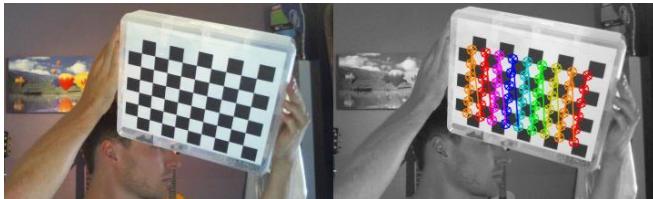


Figure 2: checkerboard pattern for camera calibration

OpenCV has a few built-in functions for calibrating camera pairs and generating depth maps [3]. These have been used to develop a built-in calibration wizard for camNode. Once the calibration has been finished, the wizard will save the parameters in a configuration file so they can be loaded when the program starts.

StereoSGBM is the fastest OpenCV – algorithm for generating depth maps. It achieves a solid 25 FPS on an Intel Core i7 – 3610QM CPU @ 2.30 GHz. However, the algorithm is far from perfect. It relies on matching blocks of pixels in both images to each other and using their coordinates to estimate the depth of that block. When a block can't be matched, or is incorrectly matched, distortion and noise appears on the depth map which will hinder the algorithm for passenger detection. The cameras used for depth map generation are two *Logitech C525* on a resolution of 640x360 pixels (max. 1280x720 supported).

### C. Network discovery

Since trains can change composition from trip to trip, it is important that the camServer and camNode instances can detect each other's presence easily on the network. ZeroConf, which stands for zero-configuration networking, is a protocol that can be used to announce a piece of software on the network with an unique name, together with a service type and some metadata in the form of key-value pairs (such as port

numbers). The announcement name for camNode will be a numeric ID which is based on the MAC – address of the first network card on the system. This way, each camNode has a unique hardware related ID that can be linked to counting results and error reports.

The C++ implementation of the protocol used in camNode is called *Servus* by *Human Brain Visualization Project* [4].

### D. Network video streaming

To stream depth images over the network, a system will be used based on the built-in socket functions from Linux. Steve Tuenkam and Isaac Maia [5] published a proof of concept for streaming OpenCV images over a local TCP network connection. The depth images will be streamed in grayscale with a fixed resolution of 640x360. This results in 230,400 bytes per image or 2.3 MB/s if video is streamed at 10 FPS. This fixed bitrate system is relatively easy to implement and causes little extra overhead for the microcomputer. However, this may put a lot of stress on the network if a lot of camNode instances are streaming simultaneously. Implementation of a video compression method and variable bitrate streaming is a subject for further research.

### E. Network command interface

In order to let camServer communicate with camNode and exchange configuration data and error reports, a separate TCP – interface is needed. The ZeroMQ library [6] was used to implement this system, since it's easy to set up and is designed to transfer short messages of variable length. With this system a camServer can send commands to the camNode to start generation of depth maps, switch to a normal RGB camera feed, retrieve configuration data (such as its location in the train) or fetch the last error generated by the software. In this setup, it's always the task of the camServer to take initiative and send a command. The camNode can only respond when it has received a command.

### F. Configuration

To give the end user the ability to change the behavior of camNode, some settings and metadata were saved in a configuration file. The FileStorage class from OpenCV was used to generate, load and parse this configuration file. Currently, the configuration file consists of stereographic calibration pattern parameters (size of the checkerboard) and allows the end user to enable or disable certain features, such as the command interface, the built-in webcam and the video streaming server. Other metadata, such as the location of the camNode in the train or a wagon identification number, is currently not implemented. However, it can be added easily in further work.

### III. DEVELOPMENT OF CAMSERVER

The camServer software has X tasks: discover other camServer and camNode instances in the network, gather videofeeds from camNode instances, execute an algorithm for passenger detection and send counting results and error reports to an external wayside server.

Since the discovery, network command interface and video feed are built on the same code and principles as camNode, they won't be discussed in detail anymore.

#### A. Master-Slave system

Since one train can have multiple servers onboard, it's interesting to distribute the load of the APC – system over different camServer instances. To regulate this distribution, one camServer will be appointed as master at the start of each new trip. When a trip starts, the camServer instances will search for each other in the network. Then, the camServer with the smallest network identifier will take up the task of master camServer. When a camServer is started in master mode, it will search for all the available camNode instances and assign them equally to the other camServer instances (including itself). Once that's done, the camServer instances will wait for a signal from the existing train software to start detection. Every time a train departs from a station, the master camServer will gather the counting results from all camServer instances and save them in memory. When a trip is finished, the master will send the results to the wayside server. Finally all camServer instances will reset and await the initialization of a new trip to repeat the whole process.

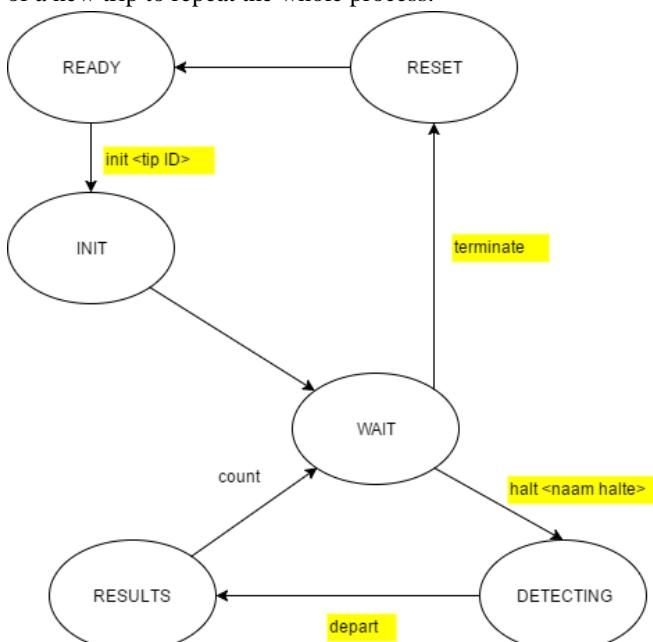


Figure 3: camServer represented as a state chart. Trainside commands marked in yellow. Master commands marked in white.

#### B. Communciation with trainside

Initiating a new trip, signaling when the train halts and departs at a station and terminating the trip are tasks of the existing trainside software. It is assumed that this software has knowledge about the current trip (identification number, a set of stations linked to this trip, the state of the current train and the name of the current station).

Each trainside server will run an instance of camServer next to the existing software. Of course the existing software will have to be modified to be able to communicate with the camServer instance on its machine. For this purpose, a set of C++ interface objects have been developed to easily send commands to the camServer instance on the local machine.

#### C. Communication with wayside

An implementation of the cURL protocol, named *curlpp* for C++ [9], has been used to push the counting results and error reports to an external wayside server. Once a trip has been completed, the master camServer will execute an HTTP POST request to a PHP script on the wayside server. This request contains the counting results in serialized form, the ID of the counting session (randomly generated at the start of the trip), the trip and trajectory identification numbers and the unique announcement ID of the master. The PHP script will parse this information and put it in a MySQL table so it can be visualized by the backend.

When an error occurs on a camServer, it can be reported to the wayside server in the same manner, even if that camServer is not running in master mode. Such error reports consist of an error code, the unique announcement ID of the camServer that reported it and the cause of the error (unique announcement ID of the hardware that caused the error, a trip ID or NULL). Communication with the wayside is always asynchronous. This means that camServer can continue working even if a result or report cannot be delivered. The cURL routines will keep trying to send the results to the server in the background. The amount of attempts and the timeouts can be configured by the end user.

#### D. Passenger detection

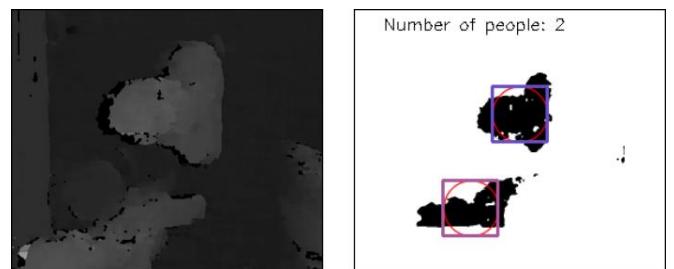


Figure 4: visualisation of passenger detection

To gain counting results, an algorithm for detecting and tracking passengers was developed. The algorithm starts by applying inverse thresholding to a depth map: all values that exceed the threshold will be set to 0, the others will be set to 1. Next, a border of positive pixels (value 1) will be made around the image, so all the negative pixels (value 0) are surrounded by positive pixels. This results in a white image with black shapes or *blobs*. Blobs are regions in an image with similarities in color and contrast. These blobs can be easily detected by the OpenCV SimpleBlobDetector algorithm. Once the blobs are detected, they are aggregated in one square region of interest (ROI). This ROI has a fixed size and contains exactly one person.

Once detection is complete, the ROI will be compared to the ROI of the detection cycle of the previous videoframe. If an overlap between a ROI of the current cycle and a ROI of the previous cycle is found, the algorithm assumes that it's the same person and their coordinates are updated. If an overlap wasn't found, the algorithm assumes a new person has walked into the frame. This person and their coordinates will be added to the tracking list.

As soon as all the new ROI have been evaluated, the entire tracking list will be checked. If there are ROI in the tracking list that haven't had an overlap for a consecutive number of times, the algorithm will assume the person has left the frame. If that person has traversed a set percentage of the frame, they will be marked as *boarded* or *departed*. Otherwise they will be discarded. The last known Y – coordinate of the person will determine if they boarded or departed the train, and their respective counter will be incremented.

### E. Watchdog

Each camServer instance has a routine, called *watchdog*, that periodically checks the health of the network. This routine checks if all camServer instances are still online, if all their assigned camNode instances are still online, if there is exactly one master in the network, if all the camServer instances have the same trip identification numbers and are in the same state. If one of the checks fails, the watchdog will attempt to recover the error and continue operation. This can happen when only a few of the camNode instances starts malfunctioning or when the current state of the camServer is different from the dominant state in the network. In any other case, the camServer will terminate itself due to an unrecoverable error.

In both cases, an error report will be sent to the wayside server.

Once a camServer has been terminated by the watchdog, the other camServer instances in the network will automatically be terminated by their own watchdogs since not all camServer instances will respond with the correct trip identifications anymore.

### F. Configuration

Just like in the camNode software, there are some parameters that can be configured by the end user. These parameters are mostly thresholds and timeouts to alter the behavior of the watchdog and cURL reporter. The URL of the wayside server can be configured in this file as well.

## IV. BACKEND

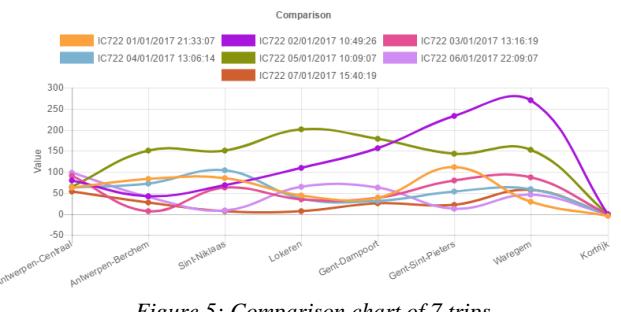


Figure 5: Comparison chart of 7 trips

Once the results and reports have been pushed to the wayside server, they can be visualized by the PHP backend. There are two main sections in the backend: trips and logs.

In the trip section, a list of trips and their status will be displayed, along with a button to view their counting results and APC logs in detail. The list will be ordered by date (recent trips first) with 50 trips per page. The list can also be filtered by date, trajectory ID or equivalent trajectory.

Individual trips can be opened from the trip section. Once a trip is opened, a full list of counting results and a graph with the number of passengers transported between two stops will be displayed. Furthermore, the APC logs from this trip, trajectory ID, trip ID and global status will be displayed.

From the trip section, a comparison wizard can be opened to compare multiple trips with each other and plot them on a graph, together with their average number of passengers and standard deviation to represent the stability of the results. Trips can be compared by trajectory ID or equivalent trajectory over a timespan or on a select set of days.

Finally, the log section of the backend will display a list of APC system logs and errors generated by the APC components. This list can also be filtered by trajectory ID, equivalent trajectory or the severity of the log entry (notice, warning or error).

## V. EVALUATION

In this section, camNode, camServer and the backend will be evaluated in terms of performance and scalability.

### A. Scalability camNode and camServer

To test scalability, a total of 40 test nodes (streaming a prerecorded video feed) were put on the network, together with a total of 1, 2 and 3 camServer instances. One emulator was put in place to send trajectory control commands, normally reserved for the existing trainside software, to the camServer instances. With this setup a trajectory with 8 stops was simulated. During the simulation, the memory usage, CPU usage and network usage were monitored using the *System Monitor* of *Ubuntu 16.04 LTS* on a computer with an Intel Core i7 – 3610QM CPU @ 2.30 GHz.

CPU and memory usage stayed acceptable throughout the simulation, with values between 5% and 15% CPU usage during detection, with memory usage between 70MB and 110MB. At the time of measurement 3 camServer instances were active.

When the load is distributed over 3 camServer instances, the network usage averages at around 35 MB/s per instance. When the load is placed on a single camServer, however, the load increases to 100 MB/s to the same machine. This puts a lot of stress on the network and comes close to the theoretical limits of a 1 Gbps network (125 MB/s). To further relieve the network, video compression will be necessary.

Process Name	User	% CPU	ID	Memory
camServer	lorre851	13	12624	70,9 MiB
camServer	lorre851	10	12285	106,9 MiB
camServer	lorre851	9	12271	87,8 MiB

Figure 6: CPU and RAM usage of three camServer instances

### B. Performance of camNode

The camNode software was tested on two microcomputers, a Raspberry Pi 3 model B and a more powerful Odroid XU4. Both microcomputers were unable to fluently run the algorithm for depth map generation and achieved speeds of only 0.4 FPS and 4FPS respectively. Streaming a single camera feed, however, worked just fine at speeds of 15 FPS and 25 FPS respectively. The conclusion from this test is that camNode can be run on a microcomputer if a single dedicated depth camera is connected.

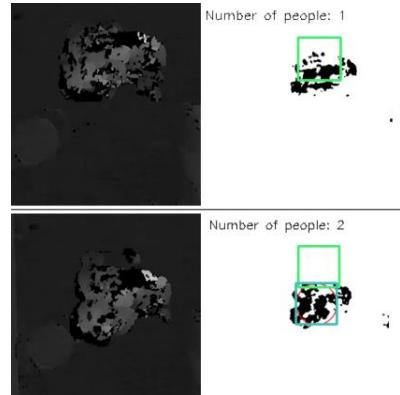


Figure 7: the passenger detection algorithm on an garbled depth map

### C. Performance of passenger detection

Due to the grave imperfections in the algorithm that generates the depth maps, it was impossible to evaluate the accuracy of the passenger detection algorithm in a meaningful way. The depth maps often contain noise and areas with incorrect values, especially when a person walks at a faster pace. These errors are due to the imperfections of the depth map algorithm and the delay between retrieving images from both cameras. Further testing of the detection algorithm with a dedicated depth camera is a subject for further research.

### D. Performance of the backend

To test the backend, the database was filled with a months' worth of counting results: 3376 daily train trips from NMBS Belgium [2], repeated 31 times, good for 1 065 035 record. To test the performance, a number of comparisons were entered into the backend while their calculation time was recorded. When the number of compared trips is limited to 15, the backend performs reasonably well with an average load time between 1 and 15 seconds. As soon as the number of trips exceeds 40, the load time increases drastically (40 seconds and above) and the resulting comparison graphs become nearly unreadable due to the amount of data points.

It can be concluded that the backend is best suited for comparing a certain equivalent trajectory during the day (trajectories which have the same stops), an individual trajectory (a trajectory on a certain time of the day) during the week or on several set dates.

## VI. CONCLUSION AND FURTHER WORK

The APC system that was developed for this thesis offers solutions for various problems linked to working with train infrastructure. By handling the counting mechanism on a trip – to – trip basis and resetting the combination of camNode and camServer instances after every trip, the problem of dynamically compiled trains has been solved. By distributing

the load of the detection algorithm over several camServer instances, the network load can be significantly reduced. Finally, by asynchronously communicating with the wayside server and only sending small amounts of information, the problem of limited available bandwidth on trains has been solved as well.

To further relieve the network load, it will be necessary to compress the images on camNode before sending them to camServer. Adding more metadata (like the location of the camNode in the train or a wagon identification number) camServer can make more intelligent choices as to when it should enable video streaming from a certain camNode. Also, the essential parameters needed to initialize a camServer trip could be backed up to a local file, so that in case of an electrical failure or crash, the session can continue by loading the data from the file.

Finally, more features could be added to the system. It could be used for CCTV, to count passengers on a per-wagon basis or to detect mechanical errors with doors on the train.

#### REFERENCES

[1] Onderwerpen van IBCN. url: <https://www.ibcn.intec.ugent.be/sites/default/files/docs/2.%20PLATO%20masterproefonderwerpen%20AJ1617%20IBCN.pdf> (visited 04-10-2016).

- [2] BeLuxTrains in het nederland. url: <http://www.beluxtrains.net/indexnl.php?page=displaybeltrains> (visited 28-04-2017).
- [3] opencv/opencv. GitHub. url: <https://github.com/opencv/opencv> (visited 09-05-2017).
- [4] Servus: Main Page. url: <https://hbpvis.github.io/Servus-1.4/index.html> (visited 20-03-2017).
- [5] Multiple streaming in c++ using opencv; OpenCV video streaming over TCP/IP. Gist. url: <https://gist.github.com/Tryptich/2a15909e384b582c51b5> (visited 30-11-2016).
- [6] Distributed Messaging - zeromq. url: <http://zeromq.org/> (visited 30-03-2017).
- [7] John WH Smith. Capturing a webcam stream using v4l2. JWH Smith. 3 dec 2014. url: <https://jwhsmith.net/2014/12/capturing-a-webcam-stream-using-v4l2/> (visited 30-11-2016).
- [8] opencv: Open Source Computer Vision Library. original-date: 2012-07-19T09:40:17Z. 10 nov 2016. url: <https://github.com/opencv/opencv>.
- [9] curlpp by jpbarrette. url: <http://www.curlpp.org/> (visited 01-06-2017).

# Inhoudsopgave

<b>Woord vooraf</b>	<b>iv</b>
<b>Toelating tot bruikleen</b>	<b>v</b>
<b>Abstract</b>	<b>vi</b>
<b>Lijst van afkortingen en symbolen</b>	<b>xviii</b>
<b>1 Inleiding</b>	<b>1</b>
1.1 Probleemstelling . . . . .	1
1.2 Vereisten Automated Passenger Counting . . . . .	2
1.3 Scope . . . . .	3
<b>2 Onderzoek</b>	<b>4</b>
2.1 Bestaande APC - systemen . . . . .	4
2.2 Hardware . . . . .	5
2.2.1 Stereografische camera's . . . . .	5
2.2.2 microcomputers . . . . .	6
2.3 Software . . . . .	6
2.3.1 Programmeertaal . . . . .	6
2.3.2 Computervisie . . . . .	7
2.3.3 Discovery . . . . .	7
2.3.4 Videostreaming . . . . .	7
2.3.5 Communicatie . . . . .	8
<b>3 Overzicht componenten</b>	<b>9</b>
3.1 Huidige situatie . . . . .	9
3.2 Uitbreidingen voor APC . . . . .	9
<b>4 Ontwikkeling camNode</b>	<b>12</b>
4.1 Detectie van hardware . . . . .	13
4.2 Stereovisie . . . . .	15
4.2.1 Cameracalibratie . . . . .	15
4.2.2 Realtime genereren van de dieptemap . . . . .	18
4.3 Netwerkinterface . . . . .	19
4.3.1 Discovery . . . . .	20
4.3.2 Command interface . . . . .	21
4.3.3 Videostream . . . . .	24
4.4 Configuratie . . . . .	26

<b>5 Ontwikkeling camServer</b>	<b>28</b>
5.1 Werkingsprincipe . . . . .	28
5.2 Netwerkinterface . . . . .	30
5.2.1 Discovery en initialisatie . . . . .	30
5.2.2 Command interface . . . . .	32
5.2.3 Interfaceobjecten . . . . .	35
5.2.4 Communicatie met trainside . . . . .	35
5.2.5 Communicatie met wayside . . . . .	36
5.3 Algoritme voor persoondetectie . . . . .	38
5.3.1 Werking . . . . .	38
5.3.2 Implementatie . . . . .	40
5.3.3 Beperkingen van stereovisie . . . . .	41
5.4 Watchdog . . . . .	41
5.5 Configuratie . . . . .	42
<b>6 Ontwikkeling back-end</b>	<b>45</b>
6.1 Databank . . . . .	45
6.2 Overzicht back-end . . . . .	45
6.3 Visualisatie van een rit . . . . .	46
6.4 Vergelijking van trajecten . . . . .	47
6.5 Fouttolerantie . . . . .	49
<b>7 Evaluatie</b>	<b>51</b>
7.1 Evaluatietools . . . . .	51
7.2 Schaalbaarheid camServer en camNode . . . . .	51
7.2.1 Bandbreedte . . . . .	53
7.2.2 Geheugen- en processorgebruik . . . . .	54
7.3 Schaalbaarheid back-end . . . . .	54
7.4 Microcomputers . . . . .	55
7.5 Stereovisie en persoonsdetectie . . . . .	56
<b>8 Uitbreidingsmogelijkheden en verder onderzoek</b>	<b>58</b>
8.1 Tellingen in wagons . . . . .	58
8.2 Doorrijdende treinen met gewijzigd traject / samenstelling . . . . .	59
8.3 Videostream met variabele bitrate . . . . .	60
8.4 CCTV . . . . .	60
8.5 Opvangen mechanische defecten treinstel . . . . .	60
8.6 Bepaling treinstellen over en onder capaciteit . . . . .	61
8.7 Herstel na stroomuitval of crash . . . . .	61
8.8 Uitbreiding configuratie camNode . . . . .	61
<b>9 Conclusie</b>	<b>63</b>
<b>10 Bijlagen</b>	<b>67</b>
10.1 GitHub repository . . . . .	67

# Lijst van figuren

1.1	Overzicht van mogelijke methodes voor passagierstellingen . . . . .	1
1.2	De verschillende componenten van een APC - systeem [2] . . . . .	2
2.1	Detectie van passagiers op basis van Histogram Of Gradients en Hough cirkel detectie [1] . . . . .	5
2.2	Intel RealSense R200 (links) [7] en Microsoft Kinect (rechts) [8] dieptecamera's .	5
2.3	Logitech C525 webcamera's . . . . .	6
3.1	Scherm in Nederlandse trein met trajectinformatie, aangestuurd door een <i>train-side server</i> [17] . . . . .	10
3.2	Schematische voorstelling van de interactie tussen camNode, camServer, de bestaande trainside software en de wayside server . . . . .	11
4.1	UML - diagram van de klasse camerascanner . . . . .	12
4.2	Gelijkijdige foto van een chronometer om vertraging tussen camera's te meten .	14
4.3	Schematische voorstelling van een stereografische opstelling [20] . . . . .	16
4.4	Detectie van 6x9 dambordpatroon in calibratieproces . . . . .	16
4.5	Mogelijke vormen van distorsie ten gevolge van de vorm van de lens van de camera [21] . . . . .	16
4.6	Vereiste volgorde van de camera's voor calibratie . . . . .	17
4.7	Twee invoerbeelden na rechttrekken en verplaatsen . . . . .	18
4.8	Voorbeeld van overeenkomstige punten met dezelfde Y-coördinaat in twee afbeeldingen [25] . . . . .	18
4.9	StereoBM dieptebeeld in 720p . . . . .	19
4.10	StereoSGBM dieptebeeld in 360p . . . . .	20
4.11	UML - diagram van de klasse cvserver . . . . .	24
5.1	Schematische voorstelling werking camServer . . . . .	28
5.2	Statendiagram camServer . . . . .	29
5.3	Flowchart werking camServer . . . . .	31
5.4	UML - diagram van de klasse iface_server . . . . .	34
5.5	UML - diagram van de klasse iface_node . . . . .	34
5.6	NMBS - traject IC 710 van Antwerpen-Centraal tot Poperinge [27] . . . . .	36
5.7	Visualisatie van persoondetectie op basis van thresholding en blobdetectie (rechts) op dieptebeelden (links). . . . .	39
5.8	UML - diagram van de klasse depthpeopledetector . . . . .	40
5.9	Flowchart camServer watchdog . . . . .	43
6.1	Visualisatie van een treinrit . . . . .	47
6.2	Vergelijking van verschillende treinritten . . . . .	48

6.3	Vergelijkingswizard . . . . .	49
6.4	Logscherf met alle APC - systeemlogs . . . . .	50
7.1	<i>Trainframe emulator</i> met een lijst van de beschikbare camNode's en camServer's in het netwerk . . . . .	52
7.2	Geheugen- en processorverbruik camServer in Ubuntu System Monitor . . . . .	54
7.3	Matching- en trackingfouten ten gevolge van imperfecte dieptemap . . . . .	56
8.1	Schematische voorstelling van de posities van de uitgangsgroep (rood) en doorgangsgroep (blauw) . . . . .	58

# Lijst van tabellen

4.1	Lijst van voor de eindgebruiker beschikbare commando's voor camNode . . . . .	22
4.2	Lijst van extra TCP controlecommando's voor camNode . . . . .	23
4.3	Lijst van TCP foutcodes voor camNode . . . . .	23
4.4	Lijst van operationele modes . . . . .	23
4.5	Lijst configuratieparameters camNode . . . . .	27
5.1	Lijst commando's camServer . . . . .	33
5.2	Lokaal formaat voor opslaan van tellingsresultaten in master camServer . . . . .	37
5.3	Mogelijke rapporteerbare foutcodes en bijhorende <i>subjects</i> van camServer . . . . .	38
5.4	Lijst configuratieparameters camServer . . . . .	44
6.1	Lay-out van de MySQL - tabel <i>railrecords</i> . . . . .	45
6.2	Lay-out van de MySQL - tabel <i>railreports</i> . . . . .	46
6.3	Lay-out van de MySQL - tabel <i>railtrips</i> . . . . .	46
6.4	Statuscodes van een specifieke treinrit. . . . .	48
7.1	Lijst van testhardware voor evaluatie. . . . .	53
7.2	Laadtijden voor vergelijkingen en filters in de back-end . . . . .	55
8.1	Variabelen in <i>camServer</i> met essentiële tripinformatie. . . . .	62

# Lijst van afkortingen en symbolen

- APC: Automated Passenger Counter
- API: Application Programming Interface
- ARM: Advanced Reduced instruction set computer Machine
- CCTV: Closed Circuit TeleVision
- CPU: Central Processing Unit
- Gb / s of Gbps: Gigabit per Seconde
- GPS: Global Positioning System
- JPEG: Joint Photographic Experts Group
- IP: Internet Protocol
- NFC: Near Field Communication
- LTS: Long Term Support
- Mb / s of Mbps: Megabit per Seconde
- MB / s of MBps: Megabyte per Seconde
- NMBS: Nationale Maatschappij der Belgische Spoorwegen
- RAM: Random Access Memory
- SDK: Software Development Kit
- SGBM: SemiGlobal Block Matching
- TCP: Transmission Control Protocol
- USB: Universal Serial Bus

# Hoofdstuk 1

## Inleiding

### 1.1 Probleemstelling

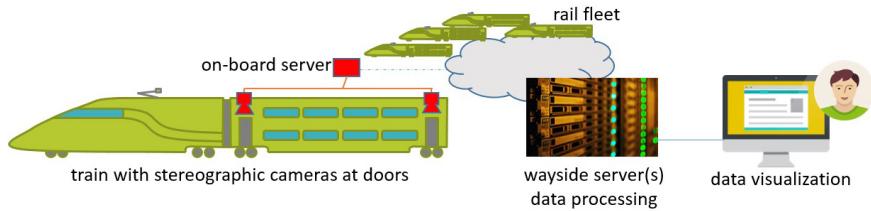
In de laatste jaren is er een stijgende vraag van treinoperatoren naar gedetailleerde gegevens over de passagiersbezetting van hun trajecten. Deze informatie staat treinoperatoren toe om de populariteit van een traject doorheen de dag in kaart te brengen en zo hun dienstregeling zo optimaal mogelijk te kunnen maken. Hiervoor zijn accurate passagierstellingen nodig op trajecten halteniveau: hoeveel passagiers er op en afstappen aan een bepaalde halte en hoeveel passagiers er vervoerd worden tussen twee opeenvolgende haltes van een traject. Dit kan tot slimme beslissingen leiden, zoals het inlassen van extra treinen op momenten dat een traject overbezet geraakt of treinen met minder capaciteit inschakelen voor minder populaire trajecten.

Er bestaan twee benaderingen voor het tellen van het aantal passagiers tijdens een rit: de manuele en de automatische benadering. Beide benaderingen splitsen zich elk op in twee methodes: de ticketafhankelijke en ticketonafhankelijke methode [1].

Ticketafhankelijke tellingen kunnen gebaseerd worden op het aantal verkochte reisbiljetten (manueel) of op het aantal gescande vervoersbewijzen bij het in- en uitstappen (automatisch). Bij dit laatste wordt het vervoersbewijs gewoonlijk voorzien van een NFC - chip en moet het vervoersbewijs bij in- en uitstappen worden gescand aan een automaat. Beide benaderingen hebben het nadeel dat ze geen rekening kunnen houden met zwartrijders. Bovendien biedt de eerste ma-

	Ticket-afhankelijk	Ticket-onafhankelijk
Manueel	Telling aantal verkochte tickets 	Manuele telling 
Automatisch	Pin pas scan 	Automated Passenger Counting 

Figuur 1.1: Overzicht van mogelijke methodes voor passagierstellingen



Figuur 1.2: De verschillende componenten van een APC - systeem [2]

nuele benadering geen inzicht in het aantal passagiers per halte, vermits er moeilijk rekening gehouden kan worden met abonnementen.

Ticketonafhankelijke systemen kunnen worden opgesplitst in manuele en automatische passagierstellingen in de trein zelf. De manuele telmethode is omslachtig (vooral tijdens piekuren) en vereist veel personeel. De tellingen zouden ook moeten gebeuren tussen twee haltes door om voldoende informatie te krijgen over het in- en uitstapgedrag van passagiers. De automatische manier, die het onderwerp vormt voor de rest van deze thesis, biedt een oplossing voor alle bovenstaande tekortkomingen. Automatische ticketonafhankelijke telsystemen tellen alle passagiers zonder tussenkomst van personeel en bieden accurate passagiersaantallen en in- en uitstapgedrag gedurende het hele traject.

## 1.2 Vereisten Automated Passenger Counting

Om automatische passagierstellingen (ook *automated passenger counting* genoemd en verder afgekort tot APC) te kunnen uitvoeren zijn drie componenten vereist [2]:

- Een sensor of camera waarmee elke in- en uitgang van het treinstel kan worden gemonitord. De gegevens van deze sensor moeten kunnen worden doorgestuurd via een lokaal netwerk naar een centrale computer.
- Een centrale computer op de trein (*traininside server*) met gespecialiseerde software die de sensorgegevens vanaf het netwerk kan verzamelen, verwerken en tellingsresultaten kan doorsturen naar een computer buiten de trein (*wayside server*).
- Een *back-end* die in staat is om de doorgestuurde gegevens te verwerken en visualiseren.

Bovendien moet een APC - systeem tegemoet kunnen komen aan de uitdagingen die horen bij het werken in een treinomgeving. Zo is er bijvoorbeeld beperkte bandbreedte met *wayside componenten* of kan de verbinding zelfs geheel wegvalLEN. Daarnaast kan de samenstelling van treinstellen van dag tot dag of zelfs binnen eenzelfde dag sterk veranderen. Een goed APC - systeem moet dus in staat zijn op een dynamische en trajectgeoriënteerde manier te kunnen communiceren met de verschillende *traininside componenten*, intelligente keuzes kunnen maken over het verwerken van sensorinformatie, opslaan en doorsturen van tellingsresultaten naar een *wayside server*, opvangen van allerhande mogelijke storingen en verwerken en visualiseren van de verzamelde gegevens in een overzichtelijke *back-end*.

Tot slot moet het systeem overweg kunnen met een zeer grote hoeveelheid data: een treinvloot kan namelijk uit duizenden treinen bestaan. Ook moeten de *traininside componenten* in staat zijn om met grote aantallen camera's overweg te kunnen. In België, bijvoorbeeld, worden

er in 2017 op weekdagen ongeveer 3600 trajecten per dag gereden met treinstellen van soms wel 10 wagons lang, wat resulteert in 40 te monitoren in- en uitgangen [3].

### 1.3 Scope

Het doel van deze thesis is het ontwikkelen van een volledig APC - systeem op basis van stereografische camera's (ook dieptecamera's genoemd) dat tegemoetkomt aan de uitdagingen die in paragraaf 1.2 werden besproken. Hierbij zal eerst onderzoek gedaan worden naar bestaande APC - systemen, technieken voor persoonsdetectie, geschikte hardware en bestaande software-pakketten om specifieke problemen mee op te lossen (zoals dynamische *discovery* van camera's en andere componenten in het netwerk, manieren om berichten en video over het netwerk te versturen en frameworks voor computervisie). Volgende zaken zullen worden onderzocht:

- De mogelijkheden van passagiersdetectie met een stereografische camera.
- De ontwikkeling van een softwarepakket dat vanop een microcomputer camerahardware kan aansturen en dieptebeelden kan streamen naar een centrale *traininside server*.
- De ontwikkeling van een softwarepakket dat op de *traininside servers* zal worden uitgevoerd. Deze software staat in voor het automatisch detecteren en aansturen van de microcomputers, ontvangen van videostreams, uitvoeren van het detectiealgoritme, communiceren met de andere boordsystemen om trajectinformatie te kunnen vergaren, tellingsresultaten doorsturen naar de *wayside server* en rapporteren van fouten die zich kunnen voordoen.
- De ontwikkeling van een *back-end* waarin gegevens gevisualiseerd kunnen worden op een betekenisvolle manier en waar foutrapporten kunnen worden bekijken.

# Hoofdstuk 2

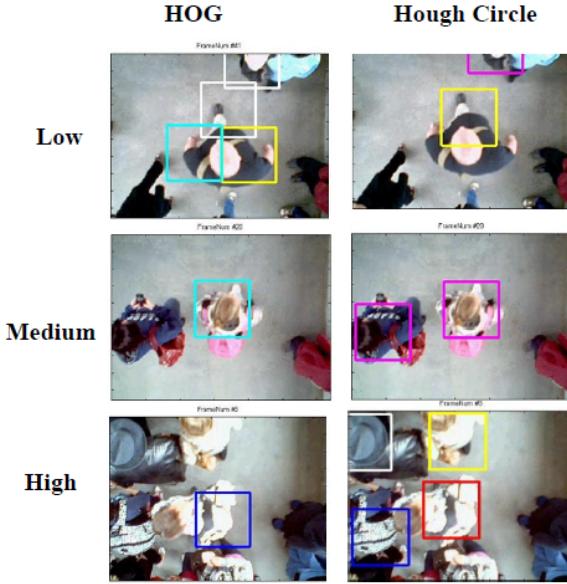
## Onderzoek

### 2.1 Bestaande APC - systemen

Om een idee te krijgen van de basisaspecten van een APC - systeem werd gekeken naar onderzoeksrapporten van Benedetto Barabino et. al [1] en Satarupa Mukherjee et. al. [4]. Het werk van Barabino is eerder gefocust op de mogelijkheden van een APC - systeem voor bussen, maar biedt wel inzichten over de huidige beschikbare tellmethodes naast APC (zie figuur 1.2) en de mogelijkheden en uitdagingen van APC - systemen. Deze zaken werden reeds in paragraaf 1.1 besproken. Vooral informatie over de passagiersbezetting per trajectsegment is een belangrijk inzicht dat APC - systemen kunnen vergaren.

Het werk van Mukherjee focust eerder op computervisie. Specifiek gaat het om een detectiesysteem voor passagiers gebaseerd op een RGB - camera vanaf het bovenaanzicht. Mukherjee beschrijft een techniek gebaseerd op het HOG (Histogram of Gradients) - detectiealgoritme en het Hough Circle detectiealgoritme. Beiden zijn interessante pistes voor het bouwen van een detectiealgoritme op basis van stereografische camera's, maar zoals later blijkt zijn andere algoritmes meer geschikt wanneer er dieptebeelden gebruikt worden in plaats van RGB - beelden.

Tot slot werden twee commerciële systemen bekeken. De *TM8206* van *Zhengzhou Tiamaes Technology Co., Ltd.* [5] is een APC - systeem voor bussen en treinen dat bestaat uit twee componenten: een stereografische camera (gebaseerd op infraroodreflecties) met netwerkinterface en een centrale verwerkingseenheid. Dit systeem heeft een geavanceerd detectiealgoritme aan boord en claimt een accuraatheid van 95 procent door onder andere schouder-aan-schouderdetectie, bagagedetectie en statistische analyses. De *Watchdog* van *Shenzhen Auto Watchdog Electronics CO., Ltd.* [6] gebruikt daarentegen een stereografische camera met twee RGB - camera's. De camera's worden via een analoge composietverbinding aangesloten op een centrale node die het tellingsalgoritme zal uitvoeren. Dit laatste systeem is eerder bedoeld voor tellingen in bussen, wat het aantal camera's per voertuig drastisch vermindert en het gebruik van analoge videosignalen mogelijk maakt. Het tellingsalgoritme haalt hier een precisie van 98 procent. De centrale verwerkingseenheid bevat hier ook zaken zoals ingebouwde 3G - connectiviteit, ingebouwde GPS en een open SDK. Prijzen voor beide systemen zijn echter niet terug te vinden.



Figuur 2.1: Detectie van passagiers op basis van Histogram Of Gradients en Hough cirkeldetectie [1]



Figuur 2.2: Intel RealSense R200 (links) [7] en Microsoft Kinect (rechts) [8] dieptecamera's

## 2.2 Hardware

### 2.2.1 Stereografische camera's

Bij de start van het project werd reeds beslist om zelf een stereografisch systeem te ontwikkelen op basis van twee identieke webcamera's. Dergelijke hardware is relatief goedkoop, makkelijk te verkrijgen en voldoende om de mogelijkheden van passagiersdetectie op basis van dieptebeelden verder te onderzoeken. Het nadeel is dat het extra implementatiewerk vergt om de twee afzonderlijke videobeelden tot één dieptebeeld. Voor deze implementatie werden twee *Logitech C525* webcamera's gebruikt.

Naast deze webcamera's werd ook onderzocht of het mogelijk was om een *Microsoft Kinect voor XBOX 360* te gebruiken. Dit apparaat bevat een 640x480 RGB - camera en een afzonderlijke stereografische camera op dezelfde resolutie die dieptebeelden kan genereren met behulp van een infraroodlaser [8]. Bovendien zou het in theorie mogelijk moeten zijn om dergelijke hardware met OpenCV aan te sturen op dezelfde manier als standaard webcamera's. De hardware werkend krijgen op een Linux besturingssysteem bleek echter lastig, vermits het stuurprogrammapakket en de uitgebreide API van Microsoft niet beschikbaar zijn voor Linux. Er bestaat wel



Figuur 2.3: Logitech C525 webcam's

een *third party driver* [9] die kan worden gebruikt in combinatie met het framework *OpenNI2* [10], maar die leek niet in staat de hardware (correct) te kunnen bedienen op *Ubuntu 16.04 LTS*. Aangezien een stabiele werking hiermee niet kon worden gegarandeerd werd deze piste verder niet onderzocht. Een implementatie van een APC - systeem voor *Microsoft Windows* maken was bovendien niet mogelijk vermits de reeds bestaande *trainside servers* doorgaans een variant van Linux draaien als besturingssysteem.

Een betere kandidaat voor verder onderzoek en ontwikkeling is de *Intel RealSense R200*. Naast een 1080p RGB - camera kan dit toestel ook dieptebeelden genereren met behulp van twee infraroodcamera's op een resolutie van 640x480 pixels en heeft een bereik van 51cm tot 400cm [7]. In tegenstelling tot de Kinect wordt de RealSense wél ondersteund voor Linux door de fabrikant. Daarnaast maakt zijn compactheid (130mm x 20mm x 7mm) en prijs (169 dollar) [11] zeer interessant voor verdere ontwikkeling of zelfs tests in een reële omgeving.

### 2.2.2 microcomputers

Om dieptecamera's te kunnen aansturen via het netwerk zal een microcomputer gebruikt worden als tussenschakel. Hiervoor zullen de mogelijkheden van twee ARM - gebaseerde microcomputer en een Intel - gebaseerde *thin client* worden onderzocht. De microcomputers in kwestie zijn de *Raspberry Pi 3 Model B* en de *Odroid XU4*, de *thin client* is een *Zotac ZBOXSD-ID10*. Hun specificaties zijn terug te vinden in tabel 7.1.

## 2.3 Software

### 2.3.1 Programmeertaal

Als programmeertaal werd resoluut voor C++11 gekozen. C++ heeft het voordeel performanter te zijn ten opzichte van Java (dat bovenop een virtuele machine draait) of Python (dat geïnterpreteerd wordt in plaats van gecompileerd). Performantie speelt bovendien een belangrijke rol vermits de software in staat moet zijn om in de grootste treinen 40 videostreams tegelijk binnen te nemen en te verwerken, iets wat in Java of Python meer CPU - tijd zou vereisen dan in C++. Bovendien is het perfect mogelijk om C++ te compileren en uit te voeren op Linux, net als Java of Python.

### 2.3.2 Computervisie

Om passagiersdetectie te kunnen uitvoeren is een goed framework voor computervisie noodzakelijk. Hiervoor werd meteen gekozen voor *OpenCV 3.1* [12]. Dit gratis framework voor computervisie vormt met zijn derde iteratie een zeer betrouwbare en performante basis voor de ontwikkeling van verschillende aspecten van deze thesis. OpenCV heeft een API voor het aanspreken van camerahardware (zowel RGB - camera's als dieptecamera's), genereren van dieptebepalen (stereovisie), calibratie van camera's, toepassen van allerhande beeldfilters en bevat ook verscheidene detectiealgoritmes. Daarnaast biedt het framework ondersteuning voor C, C++, Java en Python op zowel Windows als Linux. Bovendien is de hele library gratis voor onderwijsdoeleinden én commercieel gebruik.

### 2.3.3 Discovery

Vermits treinstellen continu van samenstelling veranderen zullen ook de APC - componenten continu van samenstelling veranderen. Om dit probleem op te vangen zullen bij de start van elk nieuw traject de netwerkcomponenten van het APC - systeem elkaar automatisch moeten kunnen detecteren in het netwerk. Dit proces wordt *discovery* genoemd. Hiervoor bestaan reeds tal van bibliotheken en protocollen zoals UPnP, SLP en ZeroConf. Uiteindelijk werd voor de ZeroConf implementatie *Servus* van *Human Brain Project Visualisation Software* [13] gekozen wegens zijn makkelijke implementatie in C++ en functionaliteit voor definiëren van een servicetype en de mogelijkheid tot aankondigen van een arbitrair aantal *key-valueparen* (zoals poortnummers).

### 2.3.4 Videostreaming

Vermits het aanspreken van videohardware en verwerken van videobeelden door OpenCV zal gebeuren, is er nood aan een protocol voor videostreaming dat makkelijk kan samenwerken met OpenCV. Als eerste werden de mogelijkheden tot gebruik van de *gstreamer* library [14] onderzocht. Deze library is in staat om *pipelines* op te zetten waar vervolgens videoframes naartoe kunnen worden gestuurd. De videoframes in de *pipeline* kunnen dan vervolgens door een ander toestel in het netwerk worden opgevraagd. Vermits deze *pipelines* zich gedragen als systeempadden in Linux zou het in theorie mogelijk moeten zijn om er gegevens naar toe te sturen en op te halen via een OpenCV *VideoCapture* object (zie paragraaf 4.1). In de praktijk bleek dit echter niet mogelijk: OpenCV slaagde er niet in gegevens naar de *pipeline* te sturen of eruit op te halen. Het gebrek aan overzichtelijke documentatie en handleidingen (de meeste basiscodevoorbeelden op de website van de uitgever weigerden te compileren of correct te functioneren, zelfs met de juiste libraries geïnstalleerd) werd *gstreamer* uiteindelijk aan de kant geschoven.

Als alternatief werd gekozen om de videoframes als ruwe binaire data door te sturen via *TCP - sockets*. Steve Tuenkam en Isaac Maia publiceerden in 2015 een oplossing [15] voor het streamen van OpenCV video op basis van netwerkfuncties in de Linux kernel. Deze code bestaat uit een serverapplicatie die videobeelden van de eerste aangesloten camera binnenneemt en doorstuurt over het netwerk en een clientapplicatie die (mits het juiste IP adres meegegeven wordt) de beelden ontvangt en weergeeft. Deze oplossing zal de basis vormen voor een verdere ontwikkeling van een robuust videostreamingsysteem.

### 2.3.5 Communicatie

Tot slot moeten de verschillende APC - componenten in staat zijn aansturings- en statusinfo met elkaar te communiceren. Servers moeten bijvoorbeeld aan cameranodes kunnen vragen of hun camerahardware correct functioneert, servers moeten onderling kunnen afspreken welke camera's er met welke servers gekoppeld worden en welke server de telresultaten zal opvragen en doorsturen en servers moeten aan de andere boordsystemen info over het traject kunnen opvragen (het trajectnummer, of de trein aan het rijden is, aan welke halte de trein gestopt is etc.). Hiervoor zouden in theorie dezelfde socketfuncties kunnen worden gebruikt als in de OpenCV - videotostream. Maar vermits deze functies heel wat implementatie- en configuratiewerk vereisten (wat verder wordt toegelicht in paragraaf 4.3.3) werd voor deze taak de library *ZeroMQ 4.2.1* [16] ingeschakeld. Dit is een open source library die het zenden en ontvangen van berichten van variabele lengte makkelijk maakt. Een tweede pluspunt is dat de ZeroMQ - library ontwikkeld met *thread safety* voorop, wat het zenden en ontvangen van berichten vanuit verschillende threads mogelijk maakt.

# Hoofdstuk 3

## Overzicht componenten

In dit hoofdstuk wordt een overzicht gegeven van alle hardware- en softwarecomponenten en hun onderlinge interacties. Daarna wordt in de drie opeenvolgende hoofdstukken in detail uitgelegd hoe de verschillende softwarecomponenten zijn opgebouwd.

### 3.1 Huidige situatie

Doorgaans zijn er op treinstellen enkele boordsystemen of *trainside servers* aanwezig [2]. Deze servers staan onder andere in voor bediening van informatieschermen voor reizigers en bevatten informatie over het traject. Het zou dus ideaal zijn om deze bestaande infrastructuur te gebruiken om APC - software op te draaien. Informatie over het traject, zoals trajectnummer en tussenliggende haltes, kunnen zo makkelijk gedeeld worden. Deze bordcomputers hebben typisch Linux als besturingssysteem.

### 3.2 Uitbreidingen voor APC

Naast de servers die reeds aan boord zijn zullen er twee bijkomende hardwarecomponenten worden toegevoegd in de wagons: camera's en microcomputers. Deze componenten worden *trainside componenten* genoemd. De camera's worden per één (in geval van een dieptecamera) of per twee (in geval van gewone beeldcamera's) aangesloten op één microcomputer. Deze microcomputer zal een softwarepakket uitvoeren, verder *camNode* genoemd, dat instaat voor volgende zaken:

- Detecteren van aangesloten camerahardware
- Genereren van dieptebeelden (in geval van gewone beeldcamera's)
- Aanbieden van dieptebeelden aan een *trainside server*
- Communiceren van allerhande metadata (configuratie, rapporteren van fouten, camerainformatie, positie van de camera in de wagon, ...) met een *trainside server*.

Op de trainside servers zal een softwarepakket, verder *camServer* genoemd, worden uitgevoerd dat instaat voor volgende zaken:

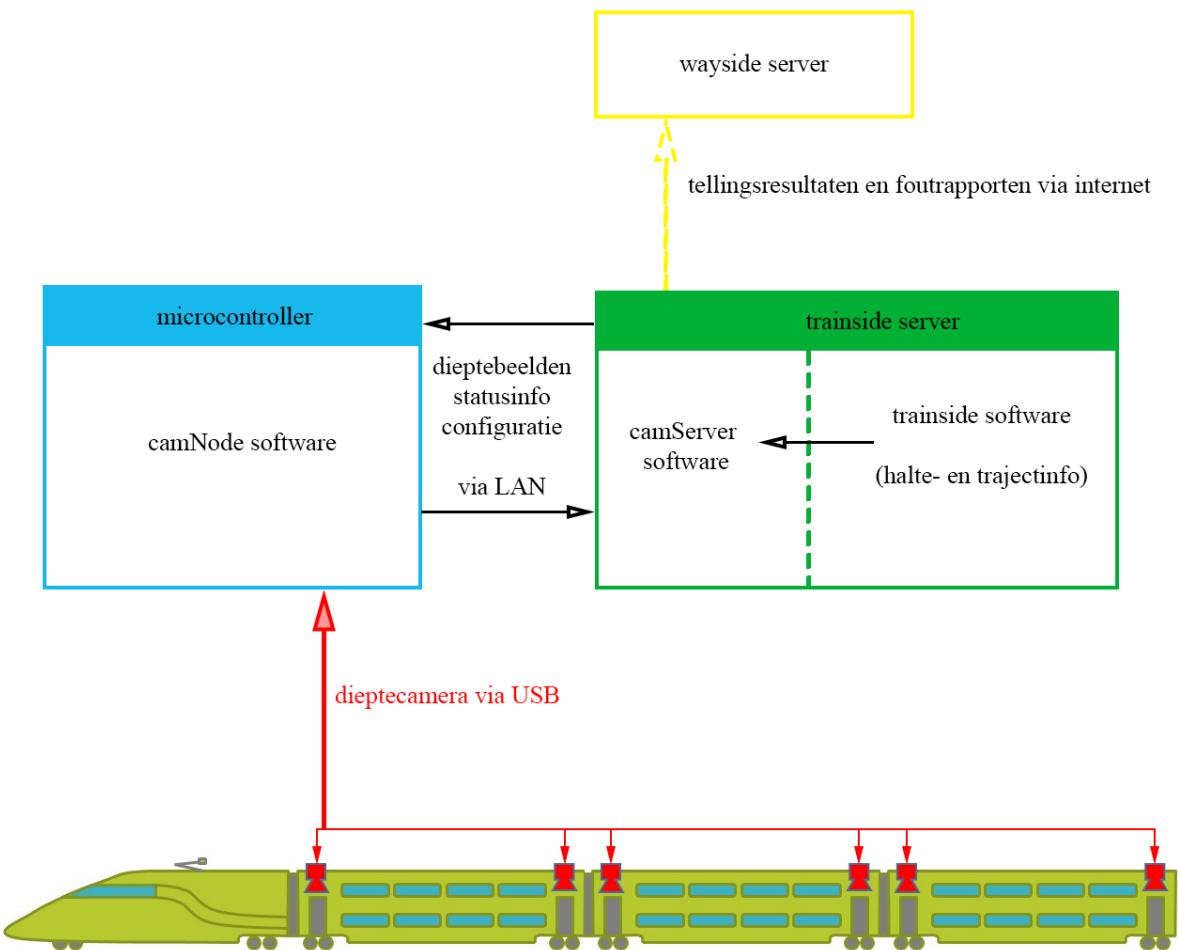
- Detecteren van camNode's in het netwerk aan de start van een traject

- Binnennemen van dieptebeelden van camNode's
- Passagiersdetectie uitvoeren op dieptebeelden
- Resultaten van passagierstellingen doorsturen naar een *wayside server*
- Communiceren met de andere software op de *trainside server* (uitwisselen van trajectinformatie)
- Rapporteren van fouten (wegvallen van componenten tijdens een treinrit, herhaaldelijk detecteren van nul passagiers op een camNode, ...) aan de *wayside server*

Tot slot zal een back-end instaan voor het visualiseren van de data en beheer van (fout)rapporten. In de komende drie hoofdstukken zal de ontwikkeling, opbouw en werking van de drie softwarecomponenten in detail besproken worden.



Figuur 3.1: Scherm in Nederlandse trein met trajectinformatie, aangestuurd door een *trainside server* [17]

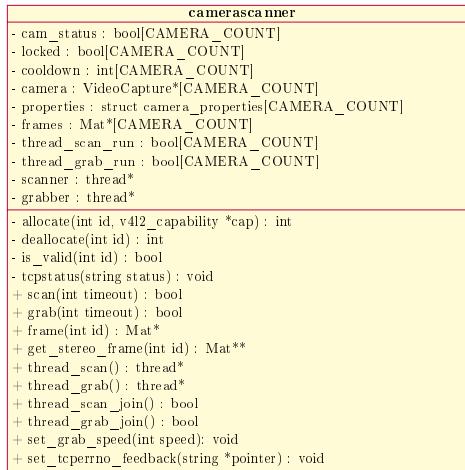


Figuur 3.2: Schematische voorstelling van de interactie tussen camNode, camServer, de bestaande trainside software en de wayside server

# Hoofdstuk 4

## Ontwikkeling camNode

In dit hoofdstuk zal de werking van het camNode softwarepakket besproken worden aan de hand van de verschillende C++ klassen die hiervoor ontwikkeld werden. Ook zullen enkele codevoorbeelden met implementaties worden toegelicht.



Figuur 4.1: UML - diagram van de klasse camerascanner

## 4.1 Detectie van hardware

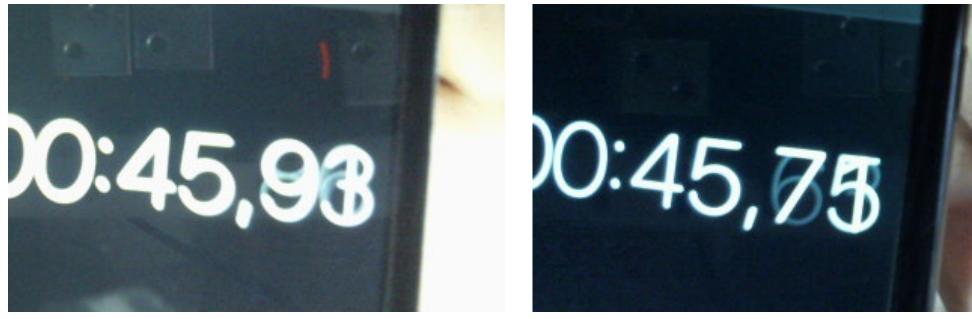
De klasse *camerascanner* staat in voor het automatisch detecteren van aangesloten camera's en het binnenvullen van beelden in een OpenCV - compatibel formaat. Het UML - diagram van deze klasse is terug te vinden in figuur 4.1.

Voor het detecteren van hardware wordt gebruikt gemaakt van de *V4L2 - library* (Video for Linux version 2). Dit is een verzameling van videodrivers gecombineerd met een API en maakt deel uit van de Linuxkernel. De implementatie van de hardwaredetectie is gebaseerd op de code en handleiding van JWH Smith [18] uit 2014. Wanneer een camera wordt aangesloten zal deze in het besturingssysteem geïdentificeerd worden met een hardwarepad in de vorm van */dev/videoX*, met X het volgnummer. Wanneer er bijvoorbeeld twee camera's worden aangesloten, zullen deze achtereenvolgens geïdentificeerd worden als */dev/video0* en */dev/video1*. Om camerahardware te detecteren wordt over de verschillende paden geïtereerd (van 0 tot een bepaalde vaste boven-grens) en gecontroleerd of er een camera op dat pad aanspreekbaar is. Dit proces kan worden geïllustreerd aan de hand van onderstaande code. In de eindimplementatie wordt dit algoritme in een lus uitgevoerd in een afzonderlijke thread (*scanner* in het UML - diagram).

```
1 // itereer over de verschillende paden /dev/videoX
2 for (int camera_count = 0; camera_count < MAX_CAMERA_COUNT; camera_count++) {
3     string index = "/dev/video" + to_string(camera_count);
4     int fd;           // file descriptor voor V4L2 - functies 'open' en 'ioctl'
5
6     // als het apparaat niet geopend kan worden, markeer als 'niet aangesloten'
7     if ((fd = open(index.c_str(), O_RDWR)) < 0) deallocate(camera_count);
8     // als het geopend kan worden, check of er metadata kan worden opgevraagd.
9     // lukt dit, markeer het apparaat als 'aangesloten en beschikbaar'
10    else {
11        if (ioctl(fd, VIDIOC_QUERYCAP, &cap) < 0) deallocate(camera_count);
12        else allocate(camera_count);
13        close(fd);
14    }
15 }
```

De status van de aangesloten camera's wordt globaal in een array van booleans opgeslagen (*cam\_status[]* in het UML - diagram), waarin de index in de array overeenkomt met de index van het hardwarepad en worden geïnitialiseerd door de functies *allocate* en *deallocate*. Deze functies zorgen er ook voor dat er een OpenCV VideoCapture - object wordt geïnitialiseerd in de array *camera[]*, langswaar er beelden van de camera's kunnen worden opgevraagd (zie verder). Een nadeel aan deze techniek is dat er op voorhand een boven-grens zal moeten worden gedefinieerd voor het maximaal aantal te detecteren camera's. Vermits dit systeem ontworpen wordt om één dieptecamera of twee standaardcamera's aan te sluiten is een vaste boven-grens niet bepaald een probleem. Om ruimte te laten voor uitbreiding en een oplossing te kunnen bieden aan mogelijke hardwareproblemen zal deze boven-grens op 10 worden ingesteld. Wanneer een camera bijvoorbeeld kortstondig verbinding verliest kan het zijn dat deze een nieuw, hoger pad toegewezen krijgt terwijl het oude pad nog niet vrijgegeven is door het besturingssysteem.

Voor het binnennemen van beelden worden verschillende functies uit de OpenCV - library gebruikt. Vermits het detecteren van passagiers ook zal gebeuren met OpenCV - functies die het OpenCV *Mat - object* gebruiken om afbeeldingen in op te slaan, is het logisch om de beelden meteen binnen te nemen in dit formaat. Bovendien zijn de functies voor aanspreken van video-



Figuur 4.2: Gelijktijdige foto van een chronometer om vertraging tussen camera's te meten

hardware gebruiksvriendelijk en tevens gebaseerd op de V4L2 - library. De OpenCV - functies zijn echter niet in staat om eenduidig te bepalen of een hardwarepad beschikbaar is of niet, wat voorgaand V4L2 - detectiealgoritme noodzakelijk maakt.

Het ophalen van frames gebeurt met twee afzonderlijke OpenCV functies: *grab*, een snelle functie die ruwe bytes van de camera binneneemt, en *retrieve*, een tragere functie die de binnengenoemde bytes decodeert en opslaat in een Mat - object. Vermits voor deze opstelling twee standaard beeldcamera's gebruikt worden is het belangrijk om de vertraging tussen de twee camera's zo klein mogelijk te maken om een correct dieptebeeld te kunnen genereren. Daarom wordt eerst achtereenvolgens de grab - functie gebruikt om bytes binnen te halen van de twee camera's, waarna ze met de retrieve - functie worden gedecodeerd. Het binnennemen van frames gebeurt hoofdzakelijk asynchroon: frames van alle aangesloten camera's worden continu opgehaald met de grab - functie in een afzonderlijke thread (*grabber* in het UML - diagram) en vervolgens met de retrieve - functie in een buffer geplaatst (*frames[]* in het UML - diagram) zodat er ten allen tijde frames beschikbaar zijn voor gebruik. Dit maakt het eenvoudiger om via de camerascanner API onmiddellijk frames te kunnen opvragen met één publieke functie (*Mat\* frame(int)* in het UML - diagram) terwijl de vertraging tussen het ophalen van beelden van alle camera's zo klein mogelijk gehouden wordt. Ondanks het gebruik van deze snelle functie blijft er een significante vertraging van 180ms tot 200ms tussen het ophalen van beelden van beide camera's, wat later voor problemen zal zorgen bij het genereren van dieptebeelden.

```

1 bool grabstatus[ CAMERA_COUNT ];
2
3 //Stadium 1: grab frames
4 for (int camcount = 0; camcount < MAX_CAMERA_COUNT; camcount++) {
5     //controleer of deze camera aangesloten is
6     if (cam_status[camcount]) {
7         //probeer een grab() uit te voeren,
8         //rapporteer of dit gelukt is
9         if (camera[camcount] -> grab()) grabstatus[camcount] = true;
10        else grabstatus[camcount] = false;
11    }
12 }
13 }
14
15 //Stadium 2: decodeer frames
16 for (int camcount = 0; camcount < MAX_CAMERA_COUNT; camcount++) {
17     while(locked[camcount]); //wacht tot de buffer vrijgegeven is
18     locked[camcount] = true; //zet buffer voor camera vast
19     //als de camera aangesloten is en grab() gelukt is,
20     //roep retrieve() aan en sla op in buffer

```

```

21     if (cam_status[camcount] && grabstatus[camcount]) {
22         camera[camcount]->retrieve(*frames[camcount]);
23     }
24     else {
25         delete frames[camcount];
26         frames[camcount] = NULL;
27     }
28     locked[camcount] = false;      // geef buffer vrij
29 }
```

In theorie zou het mogelijk sneller zijn om voor elke camera een aparte thread te voorzien om frames in op te halen. Deze aanpak werkte echter niet en leidde tot continue crashes, foutmeldingen van onderliggende V4L2 - functies en algemeen onvoorspelbaar gedrag aangezien de OpenCV - implementatie voor het ophalen van frames niet *thread safe* ontworpen is.

Een belangrijke hardwarevereiste is dat de microcomputer waarop camNode draait over evenveel afzonderlijke USB - bussen als camera's moet beschikken wanneer de camerabeelden in FFmpeg - formaat worden binnengenomen (wat hier het geval is). Worden twee camera's op dezelfde USB - bus aangesloten, dan zal het ophalen van frames voor één van de twee mislukken en wordt de foutmelding *V4L2: No space left on device* weergegeven.

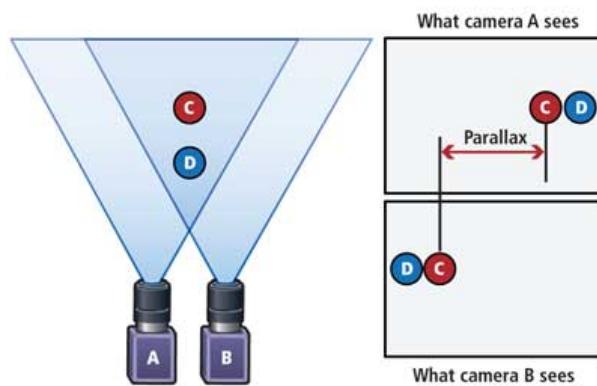
## 4.2 Stereovisie

Zoals in paragraaf 2.1 reeds werd aangehaald lijken passagierstellingen relatief makkelijk uit te voeren vanuit bovenaanzicht. Om de detectie accurater te maken wordt nu gebruikt gemaakt van een cameraopstelling met dieptezicht. Specifiek worden de mogelijkheden van een opstelling met twee identieke standaard beeldcamera's in combinatie met de principes van stereovisie onderzocht.

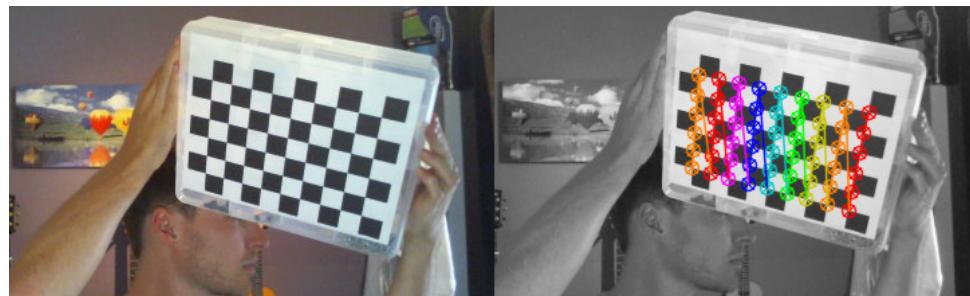
Stereovisie is het proces waarin 3D - informatie wordt vergaard uit twee 2D - beelden. Hierbij wordt een scène bekeken vanuit twee licht verschillende camerastandpunten, wat resulteert in twee quasi identieke foto's van eenzelfde scène, zoals geïllustreerd wordt in afbeelding 4.3. Wanneer camera's A en B een foto nemen van de scène, zullen punten C en D in beide foto's voorkomen, maar hun positie zal in beide foto's licht verschillend zijn. Deze afstand in overeenkomstige punten wordt *pallax* genoemd en kan worden gebruikt om 3D - informatie te vergaren [19].

### 4.2.1 Cameracalibratie

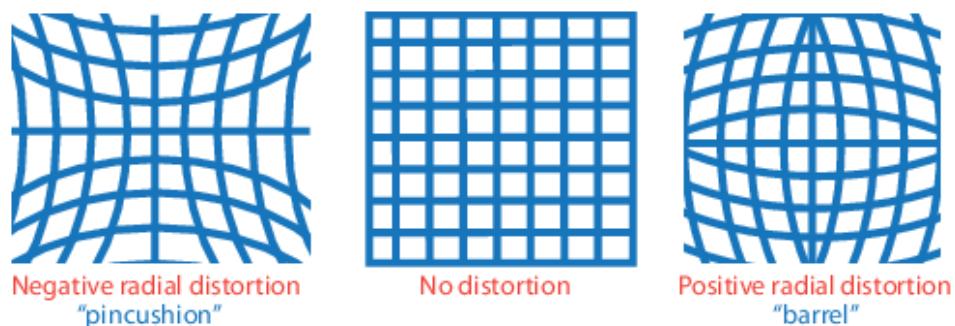
Om een dieptemap te kunnen berekenen uit twee 2D - beelden moeten de intrinsieke-, extrensieke en distorsieparameters van het camerapaar gekend zijn. Deze parameters bevatten zaken zoals de brandpuntsafstand, tilt van de camera's, afstand ten opzichte van elkaar en hoeveelheid vervorming ten gevolge van de vorm van de lens (zogenaamde *barrel* of *pin cushion* distorsie, zie afbeelding 4.5) [21] [22]. Deze parameters kunnen worden berekend met behulp van een calibratie: hierbij worden door beide camera's gelijktijdig foto's gemaakt van een zwart-wit calibratiepatroon (zoals het dambordpatroon in figuur 4.4) met gekende afmetingen. Voor een



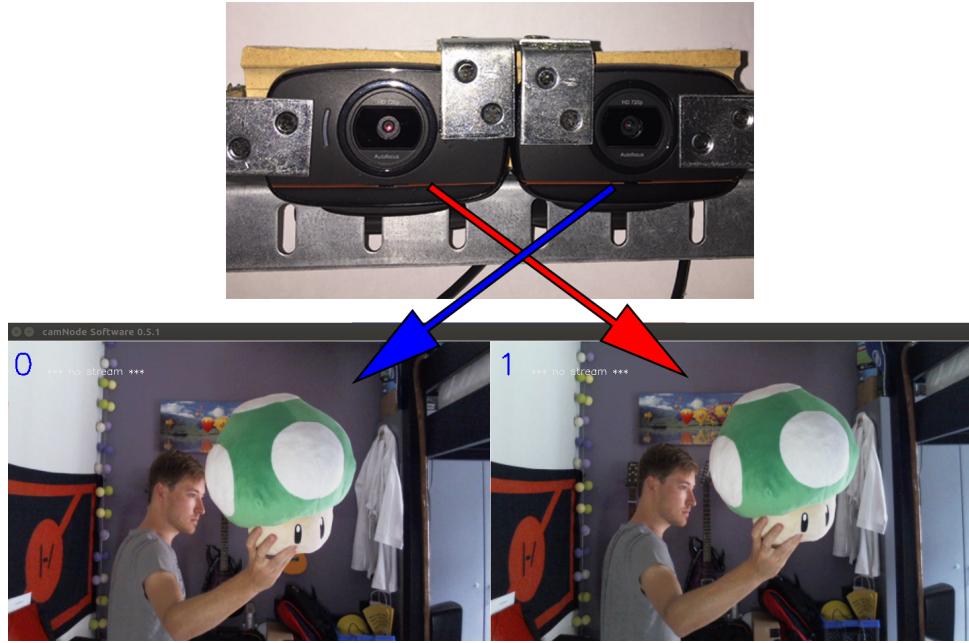
Figuur 4.3: Schematische voorstelling van een stereografische opstelling [20]



Figuur 4.4: Detectie van 6x9 dambordpatroon in calibratieproces



Figuur 4.5: Mogelijke vormen van distorsie ten gevolge van de vorm van de lens van de camera [21]



Figuur 4.6: Vereiste volgorde van de camera's voor calibratie

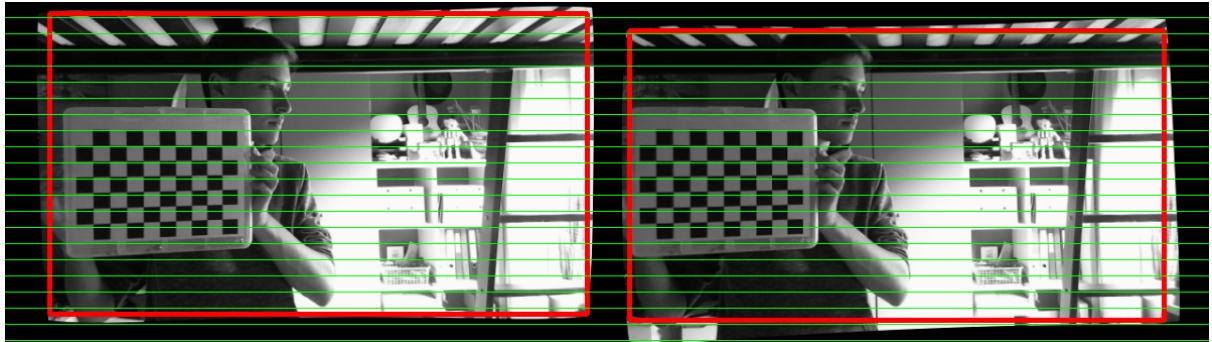
accurate calibratie moet het calibratiepatroon gefotografeerd worden in volgende posities (telkens in het volledige gezichtsveld van beide camera's) [23]:

- Van uiterst links naar uiterst rechts (X - calibratie)
- Van uiterst boven naar uiterst onder (Y - calibratie)
- Zo dicht mogelijk bij de cameraopstelling (grootte calibratie)
- Vanop verschillende afstanden ten opzichte van de camera's (grootte calibratie)
- Vanuit verschillende hoeken (hoek calibratie)

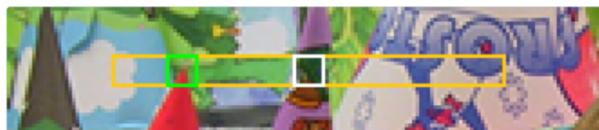
Wanneer er voldoende foto's gemaakt zijn van elke positie (in totaal een 30-tal foto's) kunnen ze gebruikt worden om de cameraparameters te berekenen.

De camNode software bevat een volledige calibratiewizard die gebouwd is op het standaard calibratieprogramma van OpenCV [24]. Het is belangrijk dat de camera's worden opgesteld zoals in figuur 4.6 aangezien de software eerst de rechtse en dan de linkse versie van de scène verwacht als inputparameters. Eens de wizard gestart is verschijnt een videofeed van beide camera's op het scherm, waarna er 30 fotoparen genomen zullen worden in intervallen van 3 seconden. Deze foto's worden opgeslagen in de werkmap, samen met een indexbestand *images.yml*. Daarna zullen de foto's door de OpenCV calibratiesoftware gebruikt worden om de cameramatrices te construeren. Aan het einde worden van deze berekening de parameters opgeslagen in twee YML - bestanden (*intrinsics.yml* en *extrinsics.yml*) en zullen de rechtgetrokken versies van de afbeeldingsparen en de gedetecteerde matches in de afbeeldingsparen op het scherm worden getoond voor visuele inspectie. Als een afbeelding niet correct gematcht blijkt te zijn kan dat afbeeldingspaar manueel uit het indexbestand verwijderd worden en kan de berekening worden herhaald zonder dit paar.

Bij elke start van camNode zal het programma eerst controleren of de bestanden *intrinsics.yml*



Figuur 4.7: Twee invoerbeelden na rechttrekken en verplaatsen



Figuur 4.8: Voorbeeld van overeenkomstige punten met dezelfde Y-coördinaat in twee afbeeldingen [25]

en *extrinsics.yml* aanwezig zijn en de calibratiedata automatisch laden indien mogelijk, zodat de calibratiewizard niet meer herhaald hoeft te worden. Het is belangrijk dat, eens gecalibreerd, de camera's relatief ten opzichte van elkaar op exact dezelfde plaats blijven staan: wanneer de posities van de camera's verandert, verandert de extrinsieke cameramatrix en moet de calibratie opnieuw worden uitgevoerd.

#### 4.2.2 Realtime genereren van de dieptemap

Om dieptebeelden te genereren moeten overeenkomstige punten in beide afbeeldingen gezocht worden. Vervolgens kan met behulp van de intrinsieke en extrinsieke parameters van de camera én de afbeeldingscoördinaten van beide punten een schatting gemaakt worden van de afstand van het punt tot de camera. Om het matchen van punten gemakkelijker te maken zullen de invoerafbeeldingen eerst rechtgetrokken worden met behulp van de informatie uit de distorsiematrix. Dit proces probeert de vervorming ten gevolge van de vorm van de cameralens op te heffen. Vervolgens zullen ze verticaal worden verschoven zodat overeenkomstige punten steeds op dezelfde Y-coördinaat liggen. Het is nu enkel nog een kwestie om voor beide punten de juiste X - coördinaat te vinden. Dit kan worden geïllustreerd aan de hand van afbeeldingen 4.7 en 4.8.

OpenCV beschikt over enkele klassen voor het genereren van dieptebeelden uit 2D - afbeeldingen: de klassen *StereoBM* en *StereoSGBM*. Beide klassen bevatten zogenaamde *block matching algoritmes*: in plaats van het matchen van individuele pixels worden telkens blokken van NxN pixels met elkaar gematcht om zo de berekeningstijd te beperken. Als maatstaf voor de overeenkomst van twee blokken wordt de *Sum of Absolute Differences* gebruikt: hierbij worden de

pixelwaarden uit een blok van de ene afbeelding afgetrokken van die uit de andere afbeelding. Vervolgens worden de waarden van de pixels bij elkaar opgeteld. Van dat resultaat wordt de absolute waarde genomen. Het blok met de kleinste absolute waarde wordt gekozen als match [25].

De tests die in dit onderdeel beschreven worden zijn uitgevoerd op een computer met een Intel Core i7-3610QM - processor. Het doel is om met algoritmes voor thresholding en blobdetectie passagierstellingen uit te voeren op deze dieptebeelden, dit wordt verder uitgelegd in paragraaf 5.3.

In een eerste poging werden dieptebeelden gegenereerd met het *stereo block matching* algoritme uit de *StereoBM* - klasse. De te matchen beelden werden in realtime van beide camera's opgehaald in een resolutie van 1280x720 pixels. Deze aanpak resulteerde echter in framerates van gemiddeld 2 frames per seconde. Daarnaast bevatten de dieptemappen zwarte ruis op plaatsen waar er geen match gevonden kon worden.

In een tweede poging werd de resolutie van de camera's gehalveerd, wat resulteerde in beelden van 640x360 pixels. Dit resulteerde in framerates van gemiddeld 10 frames per seconde. De dieptemap wordt minder scherp, maar bevat nog voldoende details om passagiersdetectie te kunnen implementeren.

In een laatste poging werd het *semiglobal block matching* algoritme uit de klasse *StereoSGBM* gebruikt. Niet alleen ging de kwaliteit van de dieptemap erop vooruit, ook de gemiddelde framerate steeg naar 25 frames per seconde. Het algoritme blijft echter imperfect: af en toe wordt de dieptemap nog overspoeld door ruis, wat later voor problemen zal zorgen bij het implementeren van een algoritme voor personendetetectie (zie paragraaf 5.3.1). Om deze matchingproblemen enigszins te beperken wordt een Gaussiaans filter toegepast op beide invoerbeelden om wat beeldruis te compenseren.



Figuur 4.9: StereoBM dieptebeeld in 720p

### 4.3 Netwerkinterface

Via het netwerk moet de camNode software tot drie zaken in staat zijn: het aankondigen van zijn aanwezigheid op het netwerk (*device discovery*), het kunnen reageren op commando's van



Figuur 4.10: StereoSGBM dieptebeeld in 360p

een centrale server en het streamen van dieptebeelden naar een centrale server.

#### 4.3.1 Discovery

Voor discovery wordt gebruik gemaakt van het *ZeroConf* (zero-configuration networking) protocol. Dit protocol kan worden gebruikt voor automatische detectie van services binnen een netwerk. Naast het adverteren van een servicetype en de naam van het toestel kunnen ook arbitraire parameters worden geadverteerd. Voor deze taken wordt de *Servus 1.5* library van *Human Brain Project Visualisation Software* gebruikt. Deze library maakt het zeer eenvoudig om een toestel in het netwerk aan te kondigen met een unieke naam, een servicetype en het poortnummer van de *TCP command interface*:

```

1 //definieer servicetype, hier "_camnode._tcp"
2 servus::Servus s("_camnode._tcp");
3 //kondig de poort aan voor de TCP command interface
4 s.set("tcp_port", tcp_port);
5 //kondig aan op het netwerk via een poort (hier via poort 100)
6 s.announce(100, unique_identifier);

```

De cameranodes worden aangekondigd als een uniek numeriek ID onder de servicenaam *\_camnode.\_tcp*. Deze *unique\_identifier* is bij voorkeur een vast hardware ID dat gekoppeld is aan een specifieke microcomputer en geregistreerd staat in een wayside database. Zo kunnen eventuele foutrapporten meteen uniek worden gekoppeld aan een bepaald toestel. In deze implementatie wordt een *unique\_identifier* gegenereerd op basis van het MAC - adres van de eerste netwerkkaart in het systeem. De eerste netwerkinterface kan makkelijk achterhaald worden door de Linux systeemmap */sys/class/net/* te scannen. De netwerkinterfaces worden daar als mappen weergegeven, bijvoorbeeld *eth0* (ethernet 0) of *wl0* (wireless lan 0). Vervolgens kan het MAC - adres bepaald worden door het lezen van het bestand */sys/class/net/<interface\_name>/address*. In het verkregen MAC - adres worden de ':' - karakters verwijderd en worden de letters vervangen door hun decimale ASCII - waarde zodat de *unique\_identifier* enkel nog uit cijfers bestaat. Deze laatste stap lijkt nutteloos, maar is belangrijk voor de latere werking van camServer vermits deze met dezelfde programmacode een *unique\_identifier* zal genereren (zie paragraaf 5.2.1).

De *tcp\_port* is de poort waarop de TCP command interface beschikbaar zal zijn. Via deze

TCP - poort zal de camServer software in staat zijn om de camNode aan te sturen met behulp van commando's. Deze interface wordt in paragraaf 4.3.2 verder in detail uitgelegd. Het doorsturen van beelden gebeurt vanaf een andere poort en wordt in 4.3.3 verder uitgelegd.

### 4.3.2 Command interface

Om camServer en camNode met elkaar te laten communiceren over het netwerk is er nood aan een manier om berichten heen en weer te sturen. Deze *command interface* is gebouwd met behulp van de *ZeroMQ* library op basis van een *request - response* - verbinding, met camServer als initiatiefnemer. Dit garandeert een robuuste controle voor het aankomen van berichten vermits aan elke request een response moet worden gekoppeld bij wijze van feedback. Het nadeel aan deze aanpak is dat het de taak wordt van de camServer - software om periodiek de foutstatus van de nodes op te vragen. Bovendien is het niet mogelijk om twee afzonderlijke berichten na elkaar te zenden zonder daartussen eerst een op een antwoord te wachten.

Hieronder volgt een voorbeeld van hoe de ZeroMQ - library geïmplementeerd is:

```

1 string tcp_port = "5000";
2 //defineer het aantal i/o threads
3 //hier 1 aangezien communicatie met slechts één camServer verwacht wordt
4 zmq::context_t context(1);
5 //definieer communicatietype als 'reply'
6 //communicatiepatroon: ontvang - zend, ontvang - zend, ...
7 zmq::socket_t tcpsocket(context, ZMQ REP);
8 //bind socket aan alle lokale netwerkinterfaces
9 tcpsocket.bind("tcp://*: "+ tcp_port);
10
11 while(running) {
12     //maak message_t object om ontvangen bytes in op te slagen
13     zmq::message_t message;
14     //wacht op het ontvangen van een bericht
15     socket.recv(&message);
16     //cast bericht naar std::string
17     string command = string(static_cast<char*>(message.data()), message.size());
18
19     /*      verwerk commando      */
20
21     //maak antwoord klaar
22     string response = "ack";
23     //cast naar message_t type
24     zmq::message_t reply(str.length());
25     memcpy(reply.data(), response.c_str(), response.length());
26     //verzend antwoord
27     socket.send(reply, ZMQ_NOBLOCK);
28 }
```

Om fouten op te vangen zijn de functies *send* en *recv* en het casten van de te versturen en ontvangen data in de eindimplementatie vervangen door de functies *string s\_recv(socket\_t &socket)* en *void s\_send(string str, socket\_t &socket)* uit de zelfgeschreven hulpklassen *zeromq\_tools*. Het casten wordt hier verborgen voor de eindgebruiker. Daarnaast wordt er tot 10 maal opnieuw geprobeerd om een bericht opnieuw te zenden of te ontvangen, telkens met een timeout van 10 milliseconden, voor er een foutmelding wordt gegeven.

Commando	Beschrijving
calibrate	Start calibratiewizard: maakt 30 fotoparen, slaat ze op als .jpg - bestanden in de werkmap, maakt een indexbestand <i>images.yml</i> , berekent de intrinsieke, extrensieke en distorsieparameters en slaat deze op in de werkmap als <i>intrinsics.yml</i> en <i>extrinsics.yml</i> .
recalibrate	Bereken calibratieparamteres opnieuw aan de hand van de laatste opgeslagen fotosequentie in de werkmap.
stereo	Zet genereren van dieptebeelden aan of uit (indien twee camera's beschikbaar en parameters beschikbaar zijn)
stereoswap	Verwissel camera 0 en camera 1 (indien het dieptebeeld er erg verstoord uitziet)
cvreset	Reset alle OpenCV GUI - elementen
streamgrid	Zet GUI aan of uit. Indien aan worden de eerste twee camera's en het gegenereerde dieptebeeld weergegeven (als er twee camera's aangesloten zijn en 'stereo' aan staat)
cvsrestart	Verbreek verbinding met alle clients en herstart de videotream server
scanfast	Zet beperking voor aantal doorgestuurde frames per seconde uit
scanmed	Beperk aantal doorgestuurde frames per seconde tot 20
scanslow	Beperk aantal doorgestuurde frames per seconde tot 10
cam	Stuur gewone beelden in plaats van dieptebeelden door over het netwerk
help	Geef een lijst van commando's weer
exit	Beëindig het programma

Tabel 4.1: Lijst van voor de eindgebruiker beschikbare commando's voor camNode

De commando's voor camNode kunnen worden ingedeeld in twee categorieën: publieke commando's en TCP - commando's. Publieke commando's kunnen door de eindgebruiker rechtstreeks in het terminalvenster ingegeven worden en dienen onder andere voor het starten van de calibratiewizard en het weergeven van de GUI. Naast de publieke commando's, die opgeliist staan in tabel 4.1, zijn er ook een aantal TCP - commando's beschikbaar die enkel over de TCP - interface te versturen zijn. Deze commando's zorgen onder andere voor het opvragen van foutmeldingen (zie tabel 4.3), het opvragen van de huidige operatiemodus (zie tabel 4.4) of het opvragen van de poort voor de OpenCV videotream. Een volledige lijst van commando's is terug te vinden in tabel 4.2. Via de *TCP command interface* zijn alle commando's uit beide categorieën te gebruiken.

In tabel 4.3 is te zien dat sommige fouten als kritiek worden beschouwd. Wanneer een camNode deze fouten signaliseert zal de camServer moeten ingrijpen door de camNode als 'onbruikbaar' te markeren en de fout rapporteren aan de *wayside* server. Sommige fouten zijn niet als kritiek gemerkeerd omdat ze niet kritiek zijn (zoals CMD\_UNKNOWN of STREAMGRID\_EXCEPTION) of omdat ze automatisch een kritieke fout uitlokken op het moment dat er dieptebeelden worden opgevraagd door camServer (zoals CAMSCAN\_ERR\_ALLOCATE, CAMSCAN\_GRAB\_FAILED of CAMSCAN\_BUFFER\_EMPTY).

Commando	Beschrijving
errno	Laatste foutmelding van het systeem, zie tabel 4.3 voor een lijst van fouten. Na ontvangst van dit commando zal de interne foutstatus wijzigen naar 'ok'.
mode	Huidige modus in binaire vorm, zie tabel 4.4 voor een lijst van mogelijke modes.
ping	Heartbeat commando. Antwoordt met 'pong'.
port	Poortnummer van de OpenCV videotostream.

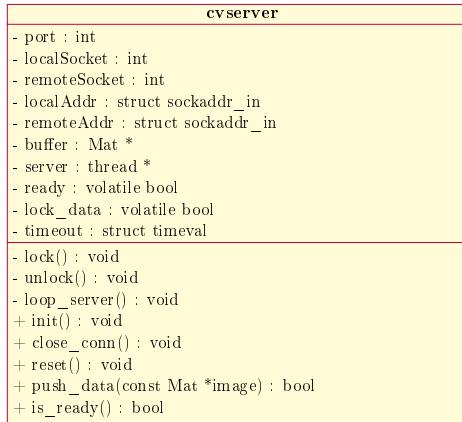
Tabel 4.2: Lijst van extra TCP controlecommando's voor camNode

Status	Beschrijving
ok	geen fouten te rapporteren
NO_CAL	[kritiek] geen stereocalibratie gevonden
STREAMGRID_EXCEPTION	onverwachte fout in de GUI
CAL_NO_STEREO	[kritiek] calibratie kon niet worden gestart wegens te weinig camera's
CMD_UNKNOWN	commando niet herkend
CAMSCAN_ERR_ALLOCATE_DEV: videoX	camera X kon niet worden gealloceerd
CAMSCAN_GRAB_FAILED: videoX	kon geen frames ophalen van camera X
CAMSCAN_BUFFER_EMPTY: videoX	kon geen frame doorsturen van camera X wegens lege buffer
CAMSCAN_NO_STEREO	[kritiek] kon geen stereoframe doorsturen omdat er geen twee frames van twee camera's beschikbaar zijn

Tabel 4.3: Lijst van TCP foutcodes voor camNode

Code	Beschrijving
1 (xxx1)	Genereren van dieptemap actief (schakelt mode 8 uit)
2 (xx1x)	Calibratiewizard actief (schakelt mode 1 en 8 uit)
4 (x1xx)	GUI actief
8 (1xxx)	Doorsturen van gewone beelden actief (schakelt mode 1 uit)

Tabel 4.4: Lijst van operationele modes



Figuur 4.11: UML - diagram van de klasse cvserver

#### 4.3.3 Videostream

Om compatibiliteit met OpenCV te behouden werd gezocht naar een oplossing om zo efficiënt en robuust mogelijk ruwe bytes over het netwerk door te sturen. Om het doorsturen van die data zo vlot mogelijk te laten verlopen wordt hier geen gebruik gemaakt van de ZeroMQ - library. In plaats daarvan wordt de low level netwerk API van Linux gebruikt in een push-pull configuratie, waarrond de klasse *cvserver* werd gebouwd. Het UML - schema van de klasse is terug te vinden in figuur 4.11. De keuze voor een *low level API* zorgt voor wat meer programmeerwerk om de sockets te initialiseren. Een korte samenvatting van de verschillende parameters en hun betekenis [26] is gegeven in onderstaand codevoorbeeld.

- AF\_INET: gebruik IPV4 (AF\_INET6 voor IPV6)
- SOCK\_STREAM: connectiegebaseerde bytestream
- SOCK\_NONBLOCK: zorg ervoor dat accept() een niet-blokkerende functie is, anders kan het programma niet worden afgesloten als er geen verbinding actief is
- MSG\_NOSIGNAL: sommige terminals beëindigen het programma met exitcode 134 als de connectie wegvalt. Deze optie verhindert dat gedrag
- SO\_REUSEADDR: maak de socket opnieuw bruikbaar na verlies van verbinding
- SO\_RCVTIMEO: timeout voor het ontvangen van een bericht ( ingesteld op 10 seconden)
- SO\_SNDTIMEO: timeout voor het zenden van een bericht ( ingesteld op 10 seconden)
- INADDR\_ANY: bind de socket met alle beschikbare netwerkinterfaces

```

1  /* Initialiseer socket      */
2  struct sockaddr_in localAddr, remoteAddr;
3  int addrLen = sizeof(struct sockaddr_in);
4  int port = 1234;
5  Mat *buffer;           // wordt asynchroon gevuld
6
7 // initialiseer socket en parameters
8 localSocket = socket(AF_INET, SOCK_STREAM | SOCK_NONBLOCK, 0);
9 setsockopt(localSocket, SOL_SOCKET, SO_REUSEADDR, &sock_opt, sizeof(int))
10 setsockopt(localSocket, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout, sizeof(timeout))

```

```

11 setsockopt(localSocket, SOL_SOCKET, SO_SNDTIMEO, (char *)&timeout, sizeof(←
12     timeout))
13 localAddr.sin_family = AF_INET;
14 localAddr.sin_addr.s_addr = INADDR_ANY;
15 localAddr.sin_port = htons(port);
16 //koppel socket aan alle netwerkinterfaces
17 bind(localSocket, (struct sockaddr *)&localAddr, sizeof(localAddr));
18
19 /* Verstuur data */
20 while(ready) {
21     //Wacht op inkomende verbinding
22     while (ready) {
23         //maak een set 'file descriptors' en voeg de lokale socket toe
24         fd_set set;
25         FD_ZERO(&set);
26         FD_SET(localSocket, &set);
27         //stel de timeout van de poller in
28         struct timeval wait;
29         wait.tv_sec = 1;
30         wait.tv_usec = 0;
31         //wacht op een inkomende verbinding in de socket
32         int rv = select(localSocket + 1, &set, NULL, NULL, &wait);
33         if (rv > 0) {
34             //accepteer de inkomende verbinding (als die er is)
35             remoteSocket = accept(localSocket, (struct sockaddr *) &remoteAddr, (←
36                 socklen_t *) &addrLen);
37             if (remoteSocket >= 0) {
38                 //verbinding succesvol
39                 break;
40             }
41         }
42         //stuur data
43         while(ready) {
44             //wacht tot de buffer vrijgegeven is
45             lock();
46             imgSize = buffer -> total() * buffer -> elemSize();           //230400 bytes
47             if ((bytes = send(remoteSocket, buffer -> data, imgSize, MSG_NOSIGNAL)) <=
48                 0) {
49                 //client heeft verbinding verbroken
50                 unlock();
51                 break;
52             }
53             //geef buffer vrij
54             unlock();
55         }
56     }
57 }

```

Bovenstaand proces wordt continu uitgevoerd bij initialiseren van de klasse *cuserver* (zie UML - schema) en draait in een afzonderlijke thread. Het aanleveren van frames gebeurt via een externe functie *push\_data(Mat\* input)* in een andere thread. De toegang tot de buffer wordt geregeld door een globale vlag van het type *volatile bool*. De functie *lock()* zal wachten tot deze vlag op *false* komt te staan, waarna ze deze zelf op *true* zal zetten en het programma verder laat gaan. Eens de bewerking met de buffer klaar is zal de vlag weer op *false* gezet worden door de functie *unlock()*.

De videofeed wordt ongecomprimeerd doorgestuurd op een voorafgesproken resolutie van 640x360 pixels. Aangezien het hoofdzakelijk om dieptebeelden gaat worden de frames in grijswaarden

doorgestuurd. Dit resulteert in 230,4kb per frame (230 400 pixels met één byte aan informatie per pixel). Om het algoritme voor personendetetectie (uitgelegd in paragraaf 5.3.1) correct te laten werken is een framerate van minimaal 10 frames per seconde vereist. Dit resulteert dus in een theoretisch netwerkverbruik van 2,3 MB per seconde per camNode.

In de huidige opstelling verwacht de ontvangstzijde telkens een beeld met vaste voorafgesproken resolutie en pixelencodering. Met andere woorden, de databursts hebben telkens een vaste voorafgesproken grootte van 230 400 bytes. Dit maakt de implementatie relatief eenvoudig maar kan het netwerk makkelijk verzadigen. Dit zou kunnen worden opgelost door toepassing van JPEG - compressie op elke frame, wat ook een apart systeem zou vereisen om telkens de grootte van de bursts op voorhand aan te kondigen. Het huidige systeem is echter weinig belastend voor de processor, wat belangrijk is wegens de beperkte rekenkracht van de microcomputers waar camNode op zou moeten kunnen draaien.

Zoals aan de commandolijst in tabel 4.2 te zien is, kan de poort van de videotostream worden opgevraagd via de *command interface*. Het is de taak van camServer om bij het initialiseren van een verbinding met camNode het IP adres van de node te achterhalen uit de *ZeroConf ID* en vervolgens de poort op te vragen via de *command interface*. Dit wordt verder beschreven in paragraaf 5.2.1.

## 4.4 Configuratie

Tot slot zijn enkele parameters (zoals poortnummers, grootte van het dambordpatroon voor calibratie en het al dan niet starten van bepaalde componenten) instelbaar door de eindgebruiker met behulp van een configuratiebestand. Bij het opstarten van camNode zal gezocht worden naar het bestand *config.yml* in de werkmap. Indien dit niet gevonden wordt zal het automatisch worden gegenereerd met standaardwaarden. Een volledige lijst van configuratiewaarden is terug te vinden in tabel 4.5. Het genereren, inlezen en bewerken van het configuratiebestand gebeurt met behulp van de OpenCV klasse *FileStorage*.

Parameter	Type	Beschrijving
checkerboard_count_height	int	Aantal kruisingen in de hoogte op het calibratiepatroon
checkerboard_count_width	int	Aantal kruisingen in de breedte op het calibratiepatroon
checkerboard_blocksize	double	Lengte van de zijde van één vierkant (in millimeter)
cvserver_port	int	Poort van de OpenCV server
tcpctrl_port	int	Poort van de command interface
start_tcpctrl	bool	Indien <i>false</i> kan camNode enkel via de lokale terminal bediend worden (bijvoorbeeld als er gecalibreerd moet worden). Indien <i>true</i> kan camNode enkel via de TCP - interface bediend worden.
start_nowebcam	bool	Indien <i>true</i> zal /dev/video0 niet worden gebruikt (bijvoorbeeld in laptops met ingebouwde webcamera).
start_cvserver	bool	Indien <i>false</i> zal de OpenCV videotostream niet worden gestart.
version	string	Versienummer van camNode. Als dit niet overeenkomt met het versienummer van de software die de configuratie laadt kan het configuratiebestand worden bijgewerkt alvorens het te laden.

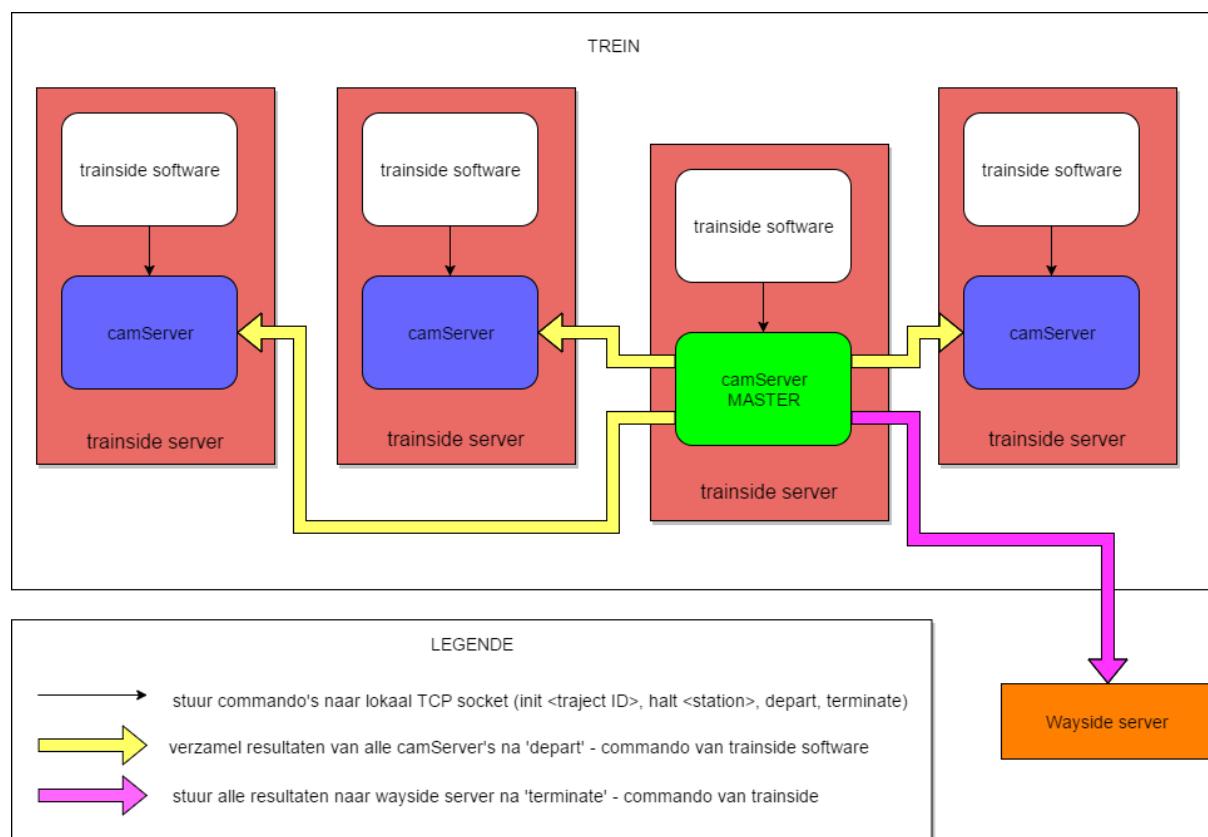
Tabel 4.5: Lijst configuratieparameters camNode

# Hoofdstuk 5

## Ontwikkeling camServer

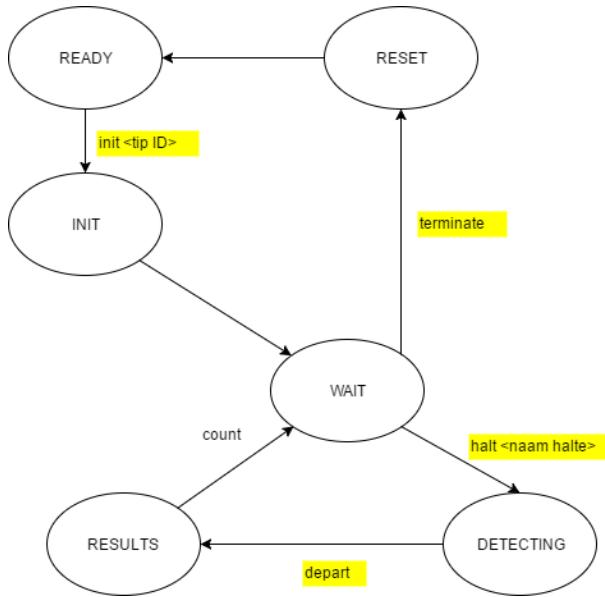
In dit onderdeel zal de werking van de *camServer software* worden uitgelegd. In paragraaf 5.1 zal het globale werkingsprincipe worden toegelicht. In de daarop volgende onderdelen zullen de verschillende taken van camServer en hun implementaties in detail worden besproken.

### 5.1 Werkingsprincipe



Figuur 5.1: Schematische voorstelling werking camServer

Een treinstel bestaat doorgaans uit verschillende wagons met enkele *trainside servers* aan boord.



Figuur 5.2: Statendiagram camServer

De bedoeling is om camServer op deze bestaande servers uit te voeren. De camServer software moet in staat zijn om te kunnen communiceren met de camNode's in het netwerk, met de bestaande software op de *trainside servers*, met andere camServer's in het netwerk en met een externe *wayside server*. Deze laatste drie interacties zijn schematisch weergegeven in figuur 5.1. Op deze afbeelding valt reeds op dat niet alle camServer's dezelfde taken hebben: één camServer zal als *master* dienen en krijgt de taak om de camNode's in het netwerk te verdelen onder alle camServer's, tellingsresultaten op te vragen van de andere camServer's en deze aan het einde van een traject door te sturen naar de *wayside server*. Daarnaast heeft deze camServer dezelfde taken als de andere camServer's, namelijk het opvragen van dieptebeelden van de camNode's en uitvoeren van een algoritme voor passagierstelling.

De werking van camServer kan als statendiagram worden beschouwd, zoals geïllustreerd in figuur 5.2. In totaal zijn er 6 verschillende staten:

- READY: er is geen traject actief, er wordt gewacht op een signaal van de *trainside software* om een nieuw traject te starten.
- INIT: de *trainside software* heeft de start van een nieuw traject aangekondigd. Er wordt in het netwerk gezocht naar andere camServer's en er wordt een *master* aangeduid. De *master* zoekt naar beschikbare camNode's en verdeelt deze evenredig over alle camServer's, vervolgens gaat deze over naar de staat WAIT. De andere camServer's gaan onmiddellijk over naar de staat WAIT.
- WAIT: de camServer's wachten op een signaal van de *trainside software*, ofwel moet worden aangegeven dat de trein gestopt is aan een station, ofwel moet worden aangegeven dat het traject beëindigd is.
- DETECTING: de *trainside software* heeft aangekondigd aan welk station de trein net gestopt is. Detectie wordt gestart op alle toegewezen camNode's. Intussen wordt gewacht op een signaal van de *trainside software* om detectie te stoppen.
- RESULTS: de *trainside software* heeft aangekondigd dat de trein het station verlaten heeft. Detectie wordt gestopt op alle toegewezen camNode's en er wordt gewacht tot de *master*

de resultaten komt opvragen alvorens weer over te gaan naar de staat WAIT.

- RESET: de *trainside software* heeft het einde van het traject aangekondigd. De *master* stuurt de tellingsresultaten door naar de *wayside server* en wist alle trajectgegevens uit het geheugen. De andere camServer's gaan onmiddelijk over tot het wissen van alle trajectgegevens. Vervolgens gaan ze automatisch over naar de staat READY.

Het hele werkingsprincipe van de camServer software is samengevat als flowchart in figuur 5.3. Daarin is te zien dat de keuze van de *master* afhankelijk is van de grootte van de ZeroConf *unique\_identifier*: de camServer met het kleinste ID krijgt de taak van *master* toegekend. Dit is een ID dat uniek is aan elke camServer instantie. Meer hierover zal worden uitgelegd in paragraaf 5.2.1. Deze keuze is makkelijk te implementeren en maakt qua extra belasting weinig uit vermits het takenpakket van de *master* zich beperkt tot het zenden en ontvangen van enkele extra TCP - berichten, wat weinig overhead met zich meebrengt. De toewijzing van camNode's aan camServer's gebeurt echter minder intelligent: er wordt gepoogd om aan elke camServer evenveel camNode's toe te wijzen, ongeacht de beschikbare processorkracht of bandbreedte op de computer waarop camServer uitgevoerd wordt. Een intelligentere verdeling op basis van de beschikbare systeembronnen op de *trainside servers* behoort tot de uitbreidingsmogelijkheden.

## 5.2 Netwerkinterface

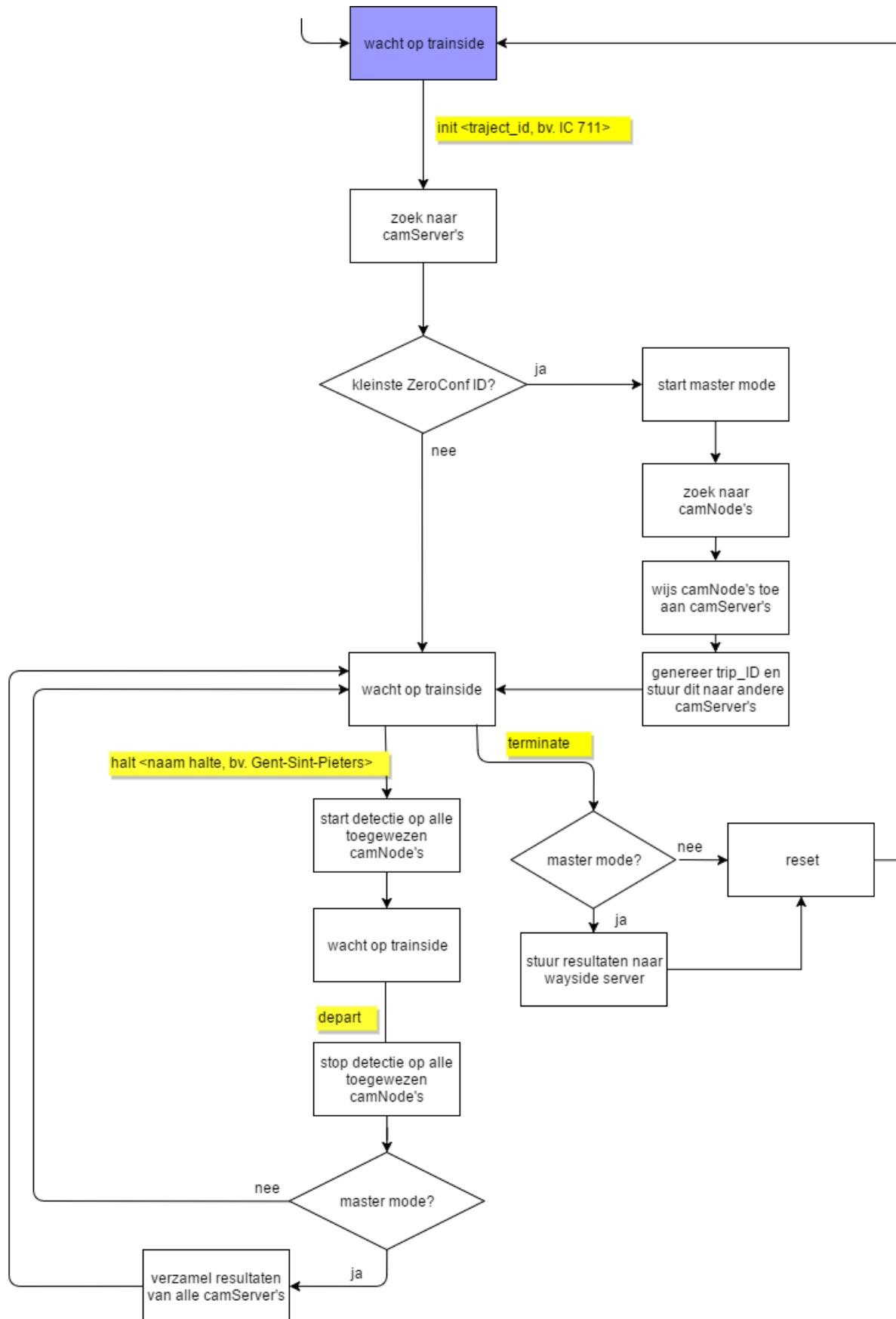
Via het netwerk moet de camServer software kunnen communiceren met andere camServer instanties, met toegewezen camNodes en met bestaande software op de *trainside servers*. Daarnaast moet de software makkelijk vindbaar zijn in het netwerk. Deze aspecten en hun implementaties worden besproken in de paragrafen van dit hoofdstuk.

### 5.2.1 Discovery en initialisatie

De discovery gebeurt opnieuw via het ZeroConf protocol. De camServer instanties zullen zich op dezelfde manier aankondigen als in paragraaf 4.3.1 werd beschreven, met een numerieke *unique\_identifier* die gegenereerd werd op basis van het MAC - adres van de eerste netwerkkaart in het systeem. De service waaronder de aankondiging gebeurt is deze keer *\_camserver.\_tcp*.

De camServer instanties moeten in de eerste plaats kunnen communiceren met elkaar. Bij het starten van een nieuwe treinrit (m.a.w. na het ontvangen van een *init* commando van de *trainside software*) zullen camServer's eerst naar elkaar zoeken:

```
1 //maak Servus object voor camserver service
2 s = new servus::Servus("_camserver._tcp");
3 //start discovery met timeout van 1000 milliseconden
4 servus::Strings camservers = s -> discover(s -> IF_ALL, 1000);
5 //itereer over alle camServer instanties
6 for( int i = 0; i < camservers.size(); i++) {
7     //converteer zeroconf naam naar hostnaam
8     string host = s -> getHost(camservers[i]);
9     //cast de aangekondigde TCP command interface poort naar int
10    int port = stoi(s -> get(camservers[i], "tcp_port"));
11    //maak interfaceobject voor camserver (host, port, unique_identifier)
12    iface_server interface(host, port, camservers[i]);
13 }
```



Figuur 5.3: Flowchart werking camServer

In het discoveryproces worden drie zaken vergaard: de *unique\_identifier* van de camServer, de hostnaam van de machine waarop camServer uitgevoerd wordt en de poort voor de TCP command interface. Met die drie gegevens kan een *iface\_server* object worden gemaakt langswaar kan worden gecommuniceerd met de camServer en statusinfo kan worden opgehaald. Het interfaceobject zal de hostnaam op de achtergrond omzetten naar het bijhorend IP - adres.

Van zodra de discovery van camServer's voltooid is zal één camServer zich als *master server* opstellen in het netwerk, namelijk de camServer met het kleinste ZeroConf ID. Deze camServer zal op dezelfde manier zoeken naar camNode's en deze vervolgens toewijzen aan de verschillende camServer's door naar hen (én naar zichzelf) een *add* commando te sturen (zie tabel 5.1). Wanneer een camServer een *add* commando ontvangt zal met de parameters 'host', 'port' en 'ZeroConf ID' een *iface\_node* object worden aangemaakt om te kunnen communiceren met de camNode. Deze interface objecten worden in paragraaf 5.2.3 verder beschreven.

Dit volledige discovery- en verdelingsproces wordt telkens éénmaal uitgevoerd bij de start van een nieuw traject, wat tegemoetkomt aan het probleem van de dynamische samenstelling van treinstellen. Er zijn echter enkele situaties waaraan dit systeem niet geheel tegemoet komt, deze worden verder toegelicht in de uitbreidingsmogelijkheden in paragraaf 8.2.

### 5.2.2 Command interface

De *command interface* voor camServer is op dezelfde manier opgebouwd als die van camNode. Er zijn in totaal vier soorten interacties mogelijk tussen de verschillende componenten in het netwerk:

- **trainside - camServer:** deze commando's dienen om camServer informatie te geven over het huidige traject en de staat van de trein (signaleren wanneer een nieuw traject begint, wanneer de trein stopt en aan welk station, wanneer de trein weer start en wanneer het traject beëindigd wordt)
- **master - camServer:** deze commando's staan in voor initialiseren van een nieuw traject, de toewijzing van camNode's aan camServer's en opvragen van detectieresultaten. Bij het begin van een traject wordt één node als master aangeduid.
- **camServer - camServer:** deze commando's worden gebruikt om de status van de andere camServer's te weten te komen en mogelijke fouten te herstellen. Deze controles worden in elke camServer uitgevoerd in een afzonderlijke *Watchdog* thread (zie paragraaf 5.4).
- **camServer - camNode:** deze commando's dienen onder andere om statusinformatie op te vragen en het genereren van dieptebeelden te starten, deze staan reeds beschreven in tabel 4.2 en tabel 4.1.

Een volledig overzicht van de mogelijke commando's uit de eerste drie categorieën is terug te vinden in tabel 5.1.

Commando	Zender	Ontvanger	Beschrijving
init /traject_id	traininside	camServer	Start een nieuwe treinrit en geef de code van het traject mee (bijvoorbeeld IC1234).
id /trip_id	master	camServer	Stelt het ID van de huidige treinrit in. Dit ID wordt bij <i>init</i> bepaald door de master en dient als unieke identificatie voor de resultaten van de treinrit.
add /host /port /unique_identifier	master	camServer	Wijst een camNode toe aan een camServer.
halt /station	traininside	camServer	Geeft aan dat de trein gestopt is aan een station. Personendetetectie zal worden gestart.
depart	traininside	camServer	Geeft aan dat de trein weer vertrekt vanuit een station. Detectie wordt gestopt.
results	master	camServer	Controleert of detectieresultaten beschikbaar zijn op alle toegewezen camNode's.
count	master	camServer	Haalt detectieresultaten op van alle toegewezen camNode's en reset alle tellers.
terminate [/force]	traininside of camServer	camServer	Beëindigt de huidige treinrit, geeft alle toegewezen camNode's vrij en wist de lijst van camServer's in het netwerk. De master zal de resultaten van de tellingen doorsturen naar de <i>wayside server</i> . Antwoordt met <i>invalid_state</i> als de <i>force</i> parameter niet is meegegeven en de camServer zich niet in de WAIT - staat bevindt.
trip	camServer	camServer	Vraagt de huidige <i>trip_ID</i> op.
current_stop	camServer	camServer	Vraagt de naam van het huidige station op. Antwoordt met <i>invalid_state</i> als de trein niet gestopt is.
master	camServer	camServer	Vraagt of de camServer in <i>master mode</i> uitgevoerd wordt.
camnodes	camServer	camServer	Vraagt hoeveel camNode's er toegewezen zijn aan deze camServer.
state	camServer	camServer	Vraagt de huidige staat van de camServer op (zie figuur 5.2).
ping	camServer	camServer	Heartbeat commando. Antwoordt met <i>pong</i> .

Tabel 5.1: Lijst commando's camServer

iface_server	
- ip : string	
- port : int	
- context : zmq::context_t *	
- socket : zmq::socket_t *	
- tcpaddr : char[100]	
- status : int	
- ID : string	
- enabled : bool	
- lock_channel : volatile bool	
+ camnodes : vector<string>	
- lock(): void	
- unlock(): void	
- reconnect(): void	
- init_socket(): void	
+ command(string command): string	
+ get_uuid(): string	
+ get_status(): int	
+ disable(): void	
+ enable(): void	
+ is_enabled(): bool	
+ iface_server(string host, int port, string unique_id)	

Figuur 5.4: UML - diagram van de klasse iface\_server

iface_node	
- ip : string	
- tcp_port : int	
- cv_port : int	
- context : zmq::context_t *	
- socket : zmq::socket_t *	
- tcpaddr : char[100]	
- status : int	
- ID : string	
- enabled : bool	
- lock_channel : volatile bool	
- detector : thread *	
- dpd : depthpeopledetector *	
- run_detection : bool	
+ camnodes : vector<string>	
- lock(): void	
- unlock(): void	
- reconnect(): void	
- init_socket(): void	
- detect(): void	
+ command(string command): string	
+ get_uuid(): string	
+ get_status(): int	
+ disable(): void	
+ enable(): void	
+ is_enabled(): bool	
+ start_cvstream(): bool	
+ stop_cvstream(): bool	
+ start_detection(): bool	
+ stop_detection(): bool	
+ is_detecting(): bool	
+ get_in(): int	
+ get_out(): int	
+ clear_results(): int	
+ iface_node(string host, int port, string unique_id)	

Figuur 5.5: UML - diagram van de klasse iface\_node

### 5.2.3 Interfaceobjecten

Om makkelijk te kunnen communiceren met camNode's en camServer's zijn er twee C++ - klassen ontwikkeld: *iface\_server* en *iface\_node*. Hun UML - schema's zijn terug te vinden in respectievelijk figuur 5.4 en 5.5. Beide objecten bevatten gelijkaardige functies voor initialisatie van een verbinding, doorsturen van commando's, opvragen van de *unique\_identifier* en verbindingsstatus. Daarnaast hebben beide interfaces de mogelijkheid om een bepaalde instantie van camNode of camServer als uitgeschakeld te markeren wanneer er fouten ontstaan in hun verbinding, werking of configuratie. Dit uitschakelen van componenten wordt hoofdzakelijk geregeld door de *watchdog* en wordt verder toegelicht in paragraaf 5.4.

Daarnaast bevat de klasse *iface\_node* ook enkele functies voor het starten en stoppen van de OpenCV videostream, het starten en stoppen van personendetetectie en het opvragen van detectieresultaten. Zaken zoals het opvragen van het poortnummer voor de videostream en de IP - adresresolutie uit de hostnaam gebeuren automatisch bij het initialiseren van het interfaceobject. Een voorbeeldimplementatie van een detectiecyclus wordt in volgend codevoorbeeld gegeven:

```
1 // initialiseer interface object
2 iface_camnode interface(host, port, zeroconf_id);
3 if(interface.get_status() == CAMNODE_CONNECTED) {
4     //wacht tot detectie gestart moet worden
5     while(state == STATE_WAIT);
6     //start detectie, wacht op het eindsignaal
7     interface.start_detection();
8     while(state == STATE_DETECTING);
9     //stop detectie, wacht tot camNode klaar is
10    interface.stop_detection();
11    while(!interface.results_available());
12    //haal resultaten op en reset tellers
13    int in += interface.get_in();
14    int out += interface.get_out();
15    interface.clear_results();
16 }
```

### 5.2.4 Communicatie met trainside

Zoals reeds in paragraaf 5.2.2 werd beschreven zijn er vier soorten interacties mogelijk op het netwerk. In deze paragraaf wordt de communicatie tussen het APC - systeem en de bestaande trainsidesoftware besproken.

In de huidige opstelling wordt verondersteld dat de trainside software in staat is om actief informatie over het traject te kunnen doorgeven aan camServer - instanties. Dit omvat het initialiseren van een nieuw traject door signalisatie van een *traject\_id* (bijvoorbeeld IC 710), aangeven wanneer de trein stopt en aan welk station, aangeven wanneer de trein weer begint te rijden en aangeven wanneer een traject voltooid is. Er wordt verondersteld dat elke trainside server waar camServer op wordt uitgevoerd kennis heeft van deze zaken. De enige uitbreiding die aan de bestaande *trainside software* moet worden toegevoegd is de mogelijkheid om deze zaken doormiddel van een TCP - commando naar de *localhost* te communiceren via de juiste poort (zie *trainside - camServer* commando's in tabel 5.1). De trainside software hoeft zich dus enkel



Figuur 5.6: NMBS - traject IC 710 van Antwerpen-Centraal tot Poperinge [27]

te ontfermen over signalisatie van een camServer instantie op zijn eigen computer. Hiervoor kan bijvoorbeeld gebruik gemaakt worden van de interfaces beschreven in paragraaf 5.2.3.

Om problemen ten gevolge van verloren of ongelijktijdige commando's op te vangen zijn enkele controles en maatregelen ontwikkeld in een zogenaamde *watchdog*. Elke instantie van camServer zal een *watchdog* uitvoeren. De werking hiervan wordt in detail uitgelegd in paragraaf 5.4.

### 5.2.5 Communicatie met wayside

Bij de start van een traject wordt één camServer als *master* aangeduid. Het is de taak van deze camServer om resultaten te verzamelen van alle andere camServer's wanneer de trein weer vertrekt aan een station en om alle resultaten door te sturen naar een externe *wayside server*. Het doorsturen gebeurt door een HTTP POST - request te sturen naar een PHP - script op de wayside server. Dit gebeurt met de cURL - library *curlpp 0.8.1* van *jpbarette* [28], een C++ wrapper voor de low level *libcurl* library. Het versturen van data gebeurt als volgt:

```

1 //maak cURL request - object klaar
2 //hier wordt het antwoord van de server in opgeslagen
3 curlpp::Easy request;
4 //stel het doel van de HTTP - request in
5 request.setOpt(new curlpp::options::Url("http://localhost/curl.php"));
6 //stel POST - variabelen in
7 curlpp::Forms formParts;
8 formParts.push_back(new curlpp::FormParts::Content("data", data));
9 formParts.push_back(new curlpp::FormParts::Content("ic", IC));
10 formParts.push_back(new curlpp::FormParts::Content("trip", trip_id));
11 //voer HTTP POST request uit
12 request.setOpt(new curlpp::options::HttpPost(formParts));
13 //bekijk het antwoord van de server
14 std::ostringstream os;
15 os << request;

```

```

16 if (os.str() == "ok") {
17     // gelukt
18 }
19 else {
20     // mislukt, probeer opnieuw.
21 }

```

In bovenstaand codevoorbeeld worden volgende variabelen doorgestuurd:

- ic: het trajectnummer of *traject\_id* (bijvoorbeeld IC 711)
- trip: het unieke ID van de huidige treinrit of *trip\_id* (gegenererd door de master camServer aan het begin van de rit)
- data: de gegevens van de tellingen geformateerd als geserialiseerde tekst. Kolommen worden gescheiden door "...", rijen worden gescheiden door ":". Het formaat waarin deze resultaten worden opgeslagen is samengevat in tabel 5.2.

Het PHP - script waar de HTTP POST request naar gestuurd wordt zal de gegevens deserialiseren en in een MySQL - tabel plaatsen waarna ze door de back-end gevisualiseerd kunnen worden.

Naast het doorsturen van tellingen kunnen ook rapporten worden doorgestuurd naar de wayside server. Deze rapporten kunnen ook door camServer's doorgestuurd worden die niet in *master mode* worden uitgevoerd. Een rapport bevat volgende parameters:

- report: de *unique\_identifier* van de zender van het rapport
- trip: het unieke ID van de huidige treinrit
- subject: de *unique\_identifier* van de veroorzaker van het incident, een trajectnummer van het actieve traject of *NULL*. Zie tabel 5.3 voor een lijst van mogelijke *subjects* per foutcode.
- error: foutcode. Een lijst van mogelijke foutcodes is terug te vinden in tabel 5.3.

Rapporten worden op dezelfde manier doorgestuurd als tellingsresultaten. Wanneer het doorsturen van een rapport of resultaat faalt (bijvoorbeeld door het wegvalLEN van de internetverbinding) zal enkele seconden gewacht worden alvorens het opnieuw te proberen. Deze *timeout* en het aantal communicatiepogingen kunnen worden geconfigureerd (zie paragraaf 5.5).

Elke communicatie met de *wayside server* gebeurt asynchroon. De camServer software hoeft dus nooit te wachten tot een rapport of resultaat succesvol verstuurd is, zodat zelfs tijdens lange perioden van verbindingproblemen de werking van het APC - systeem ongestoord verder kan gaan.

tijdstempel	naam halte	aantal personen ingestapt	aantal personen uitgestapt	aantal camnodes actief	aantal camnodes toegewezen bij <i>init</i>
-------------	------------	---------------------------	----------------------------	------------------------	--

Tabel 5.2: Lokaal formaat voor opslaan van tellingsresultaten in master camServer

error	subject	beschrijving
ERR_OK (0)	traject_id	Een nieuwe treinrit is gestart.
ERR_CAMNODE_LOST (1)	camnode_id	Een toegewezen camNode heeft onverwacht verbinding verbroken of kon niet worden bereikt.
ERR_CAMSERVER_LOST (2)	camserver_id	Een camServer kon niet meer worden bereikt.
ERR_SYNC (3)	NULL	Niet alle camServer's in het netwerk hebben hetzelfde unieke ID van de huidige treinrit.
ERR_MASTER (4)	NULL	Het aantal master camServer's op het netwerk is niet gelijk aan 1.
ERR_TERMINATE (5)	NULL	De camServer is gereset door de watchdog na een onherstelbare fout.
ERR_NO_SERVER (6)	NULL	Geen enkele camServer reageerde tijdens de initialisatie.
ERR_RESULTS_LOST (7)	NULL	De master camServer is er niet in geslaagd resultaten van alle camServer's te verzamelen of door te sturen.
ERR_STATE (8)	NULL	Niet alle camServer's bevinden zich in dezelfde staat.
ERR_CAMNODE (9)	camnode_id:errno	Er heeft zich een fout voorgedaan op een camNode. Zie tabel 4.3 voor mogelijke waarden van <i>errno</i> .
ERR_CAMNODE_CRIT (8)	camnode_id:errno	Er heeft zich een kritieke fout voorgedaan op een camNode. Zie tabel 4.3 voor mogelijke waarden van <i>errno</i> .
ERR_END (20)	traject_id	De treinrit werd succesvol beëindigd.

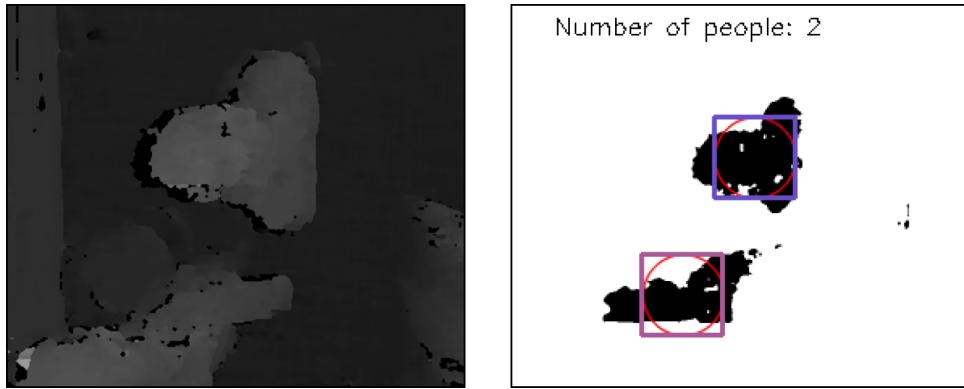
Tabel 5.3: Mogelijke rapporteerbare foutcodes en bijhorende *subjects* van camServer

### 5.3 Algoritme voor persoondetectie

Om passagierstellingen te kunnen uitvoeren moet persoonsdetectie- en tracking worden geïmplementeerd. De detectie staat in voor de effectieve tellingen van het aantal passagiers terwijl de tracking helpt om te kunnen bepalen of een passagier in of uit de wagon gestapt is. Het algoritme start vanaf een dieptebeeld. Elke pixel heeft een waarde tussen 0 en 255 (grenzen inbegrepen). Hoe hoger de waarde, hoe dichter dit punt zich bij de camera bevindt. Voor pixels met waarde 0 is geen diepteïnformatie beschikbaar.

#### 5.3.1 Werking

Als eerste stap wordt thresholding toegepast: pixels hoger dan een bepaalde waarde worden op negatief (waarde 0) ingesteld. Alle andere pixels worden op positief (waarde 1) gezet. Vervolgens wordt aan de randen van het beeld een offset van positieve pixels toegevoegd zodat alle negatieve pixels voldoende omgeven zijn door positieve pixels, wat nodig is om *blob detectie* te kunnen uitvoeren. Het resultaat is een binaire map waarin mensen als zwarte vlekken met scherpe rand worden geïdentificeerd. Dit resultaat is te zien aan de rechterkant van figuur 5.7.

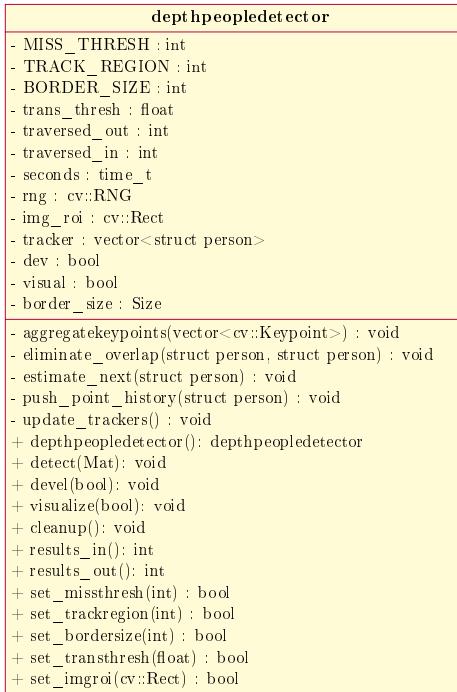


Figuur 5.7: Visualisatie van persoondetectie op basis van thresholding en blobdetectie (rechts) op dieptebeelden (links).

Op deze binaire map wordt *blobdetectie* uitgevoerd met de OpenCV - klasse *simpleBlobDetector*. *Blobs* zijn groepen pixels in een afbeelding met gelijkaardige eigenschappen zoals kleur of helderheid. Het doel van de blobdetector is om de zwarte regio's die ontstaan zijn te identificeren als *keypoints*: cirkels met x- en y-coördinaten en een straal die de blobs aanduiden. Daarna worden overlappende en nabijgelegen keypoints samengenomen tot *regions of interest*: dit zijn vierkante regio's met vaste grootte waarin zich exact één persoon bevindt.

Eens deze *regions of interest* gedefinieerd zijn worden ze vergeleken met de regions of interest van de detectie in het voorgaande frame. Wanneer één van deze regio's overlapt met een regio uit de vorige detectie gaat het om dezelfde persoon en worden de coördinaten van deze persoon bijgewerkt. Wanneer geen enkele regio overeenkomt wordt deze regio geregistreerd als nieuwe persoon. Naast de coördinaten van de laatste detectie, de coördinaten van de twee voor-gaande detecties en de waarde van de kleinste en grootste waargenomen y-coördinaat bevat een persoonsobject ook een parameter *misses* die aangeeft hoeveel keer er géén overlap gedetecteerd werd met een nieuwe region of interest uit de huidige detectieronde.

Tot slot wordt de parameter *misses* van elke persoon verhoogd met 1. Daarna wordt gecontroleerd of het aantal *misses* een bepaalde threshold overschrijdt. Wanneer dat zo is wordt gecontroleerd wat het verschil is tussen de kleinste en grootste waargenomen y - coördinaat. Is de absolute waarde van dit verschil groter dan een threshold, dan is de persoon in of uit de trein gestapt. Is de laatst waargenomen y - coördinaat groter dan de helft van de grootte van het frame, dan is de passagier de trein ingestapt en wordt de teller voor aantal ingestapte passagiers verhoogd met 1. Anders is de passagier de trein uitgestapt en wordt de teller voor aantal uitgestapte passagiers verhoogd met 1. Tot slot wordt de persoon uit de trackinglijst verwijderd.



Figuur 5.8: UML - diagram van de klasse depthpeopledetector

### 5.3.2 Implementatie

Het detectiealgoritme is ondergebracht in de klasse *depthpeopledetector*. Het UML - schema van deze klasse is te vinden in figuur 5.8. De gegevens van een persoon (laatste drie coördinaten, aantal *misses*, minimum- en maximumpositie en aantal opeenvolgende succesvolle trackingrondes) zijn ondergebracht in de struct *person*. Een voorbeeldimplementatie van de *depthpeopledetector* is te vinden in onderstaand codevoorbeeld:

```

1 //maak detectorobject aan en zet visualisatie van de detectie aan indien ←
2     gewenst
3 depthpeopledetector dpd();
4 dpd.visualize(true);
5 //maak een cvclient object en verbind met de camNode videotostream
6 cvclient client(port, ip_addr);
7 //voer detectie uit zolang camServer in de staat DETECT staat
8 while(state == STATE_DETECT) {
9     dpd.detect(*(client.get_data()));
10 }
11 //beslis of de personen die na einde van detectie nog in het frame waren ←
12     geklasseerd worden als in- of uitgestapt
13 dpd.cleanup();
14 //haal resultaten op
15 int in = dpd.results_in();
16 int out = dpd.results_out();

```

Verder zijn er nog enkele instelbare thresholds:

- set\_missthresh(int): de waarde die de parameter *misses* moet overschrijden voor een persoon als in- of uitgestapt moet worden geklasseerd.

- `set_trackregion(int)`: de breedte en hoogte in pixels van de *region of interest* die een persoon definieert.
- `set_bordersize(int)`: aantal positieve pixels dat aan de rand moet worden toegevoegd. Als negatieve pixels niet voldoende omgeven zijn door positieve pixels worden ze niet opgepikt door de *blobdetector*.
- `set_transthresh(float)`: percentage van het frame dat een persoon moet hebben overbrugd alvorens deze geklasseerd wordt als in- of uitgestapt nadat de parameter *misses* werd overschreden.
- `set_imgroi(cv::Rect)`: stel een globale *region of interest* in voor het dieptebeeld waarin de detectie moet worden uitgevoerd. Bij door stereovisie gegenereerde dieptebeelden kunnen de randen soms storing bevatten wat het detectiealgoritme in de war stuurt.

### 5.3.3 Beperkingen van stereovisie

De beelden die door de stereografische camera gegenereerd worden zijn imperfect. Zoals reeds in paragraaf 4.2.2 werd beschreven ontstaan er in elke dieptemap zwarte regio's waar geen puntmatching mogelijk is. Daarnaast komt het ook voor dat de puntmatching er volledig naast zit en er grote storing ontstaat op het beeld, wat resulteert in het detectie van te weinig (te kleine regio's) of te veel (regio's die onterecht als 'dicht bij de camera' worden beschouwd) blobs. Deze imperfecties kunnen ervoor zorgen dat de persoonstracking faalt.

Om deze imperfecties te compenseren zijn twee zaken geïmplementeerd. Om kleine storingen (losse zwarte pixels of uitschieters) op te vangen wordt er mediaanfiltering toegepast op het dieptebeeld. Daarnaast wordt er ook bewegingsinterpolatie toegepast op de trackers: wanneer de parameter *misses* van een persoon reeds waarde 1 had voor incrementatie in de laatste stap, wordt de positie van de *region of interest* toch gewijzigd. Het verschil van de voorgaande twee posities in x- en y-richting wordt bijgeteld bij de huidige positie. Er wordt dus geschat dat de persoon zich in dezelfde richting zal blijven voortbewegen als tijdens de vorige twee detectieronden.

## 5.4 Watchdog

Om de gezondheid van het camServer - netwerk te controleren en in te grijpen als er fouten ontstaan werd een *watchdog* ontwikkeld. Dit is een routine die op elke camServer instantie wordt gestart bij de initialisatie van een nieuwe treinrit. De *watchdog* voert periodiek volgende controles uit:

- 1. Controle of alle camServer's nog steeds bereikbaar zijn (met behulp van een *ping* commando)
- 2. Controle of alle toegewezen camNode's nog steeds bereikbaar zijn (met behulp van een *ping* commando) en of er zich fouten voorgedaan hebben (met het *errno* commando).
- 3. Controle of er zich exact één *master* in het netwerk bevindt.
- 4. Controle of alle camServer's hetzelfde *trip ID* hebben.
- 5. Controle of alle camServer's zich in dezelfde staat bevinden.

Elke controle heeft zijn eigen thresholds voor aantal toegelaten opeenvolgende gefaalde controles (opgelijst in tabel 5.4). Voor controle's 1, 3 en 4 zal de camServer onmiddelijk gereset worden door de watchdog van zodra de respectieve threshold overschreden wordt. Voor controle 2 zal dit pas gebeuren als een bepaald percentage van toegewezen camNode's niet langer bereikbaar is of incorrect functioneert.

Voor controle 5 zal een poging worden gedaan om de staat van de camServer te wijzigen naar de dominante staat in het netwerk. Volgende overgangen kunnen hierbij worden geforceerd door de *watchdog*:

- Overgang van WAIT naar DETECT door gemist *halt* commando van de *trainside*
- Overgang van DETECT naar RESULTS door gemist *depart* commando van de *trainside*

Verder zijn er nog andere niet-fatale onregelmatigheden mogelijk:

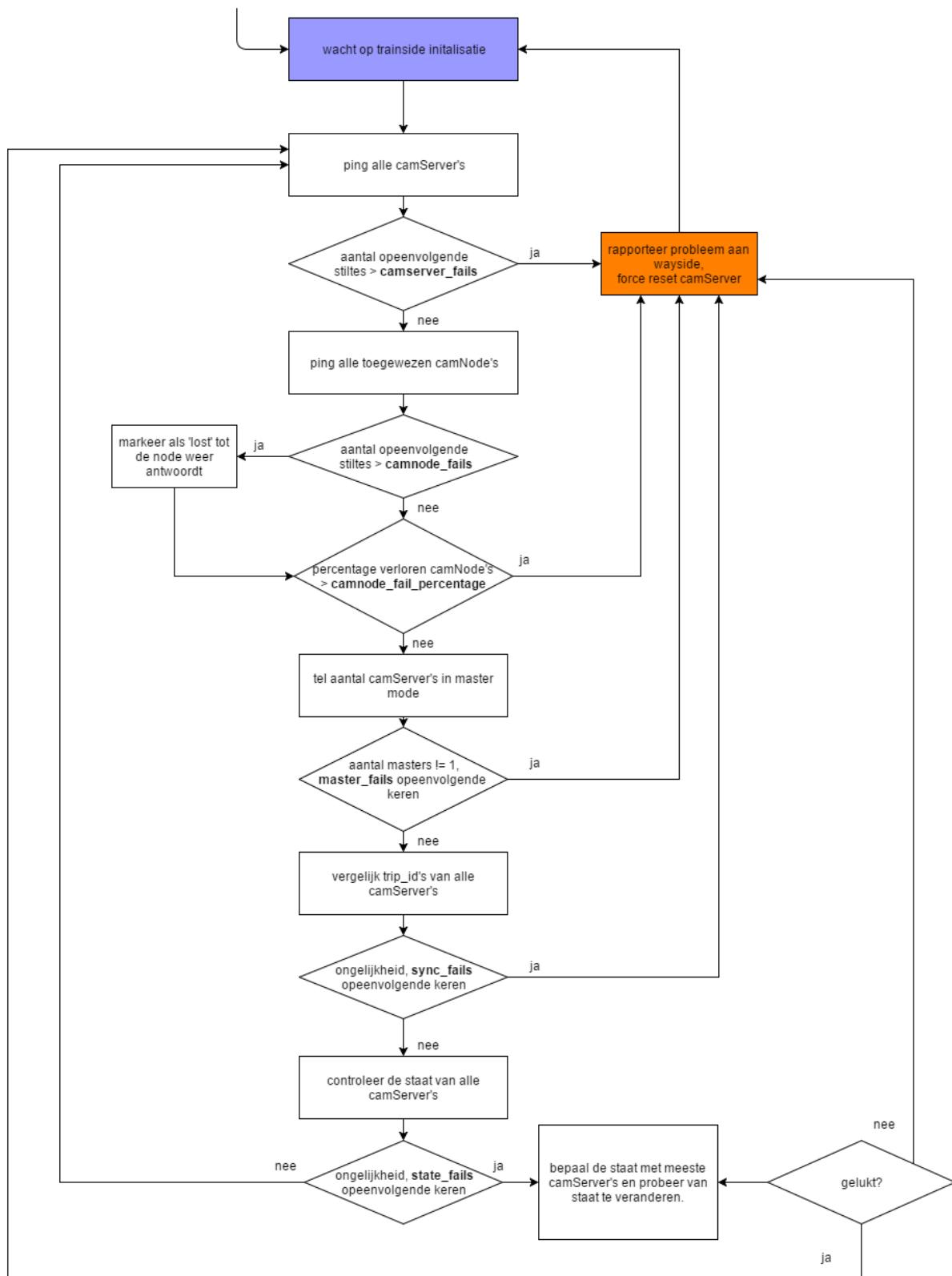
- Verloren *init* commando: wanneer een camServer een init commando niet krijgt zal deze niet kunnen deelnemen aan de verdeling van camNode's. Hierdoor zal deze camServer ook niet in de controlelijst (1) van de *watchdog* terechtkomen.
- Vast in RESET - staat tijdens initialisatie: ook hier ontstaat hetzelfde scenario als hierboven en zal de camServer niet deelnemen aan de verdeling van camNode's omdat het *init* commando wordt genegeerd in de staat RESET.

In alle andere gevallen zal de *watchdog* de camServer resetten en de fout rapporteren aan de *wayside server*. Van zodra één camServer in het netwerk gereset wordt door de *watchdog* zullen de andere camServer's in het netwerk automatisch volgen omdat controle 1 nu niet meer succesvol kan zijn.

Tot slot gaat elke reset van de *watchdog* gepaard met een foutrapportering. Een volledige lijst van mogelijke fouten is terug te vinden in tabel 5.3.

## 5.5 Configuratie

Net zoals bij camNode zijn er voor camServer enkele parameters opgeslagen in een configuratiebestand. Deze parameters zijn hoofdzakelijk thresholds die het gedrag van de *watchdog* beïnvloeden. Een volledige lijst van paramters is terug te vinden in tabel 5.4.



Figuur 5.9: Flowchart camServer watchdog

Parameter	Type	Beschrijving
gather_timeout	int	Timeout in seconden na een gefaalde poging van de <i>master</i> om resultaten op te halen van camServer's.
gather_retry	int	Aantal pogingen dat <i>master</i> mag ondernemen om resultaten van camServer's op te halen.
curl_timeout	int	Timeout in seconden na een gefaalde poging van de <i>master</i> om te communiceren met de <i>wayside server</i> .
curl_attempts	int	Aantal pogingen dat <i>master</i> mag ondernemen om te communiceren met de <i>wayside server</i> .
camnode_fails	int	Aantal keer dat een camNode achtereenvolgens niet mag antwoorden op het heartbeat commando alvorens deze als 'niet langer verbonden' beschouwd wordt.
camnode_fail_percentage	double	Percentage van toegewezen camNode's dat mag uitvallen alvorens de resultaten ongeldig verklaard worden.
camserver_fails	int	Aantal keer dat een camNode achtereenvolgens niet mag antwoorden op het heartbeat commando alvorens deze als 'niet langer verbonden' beschouwd wordt.
master_fails	int	Aantal opeenvolgende keren dat er een incorrect aantal <i>masters</i> op het netwerk mogen verschijnen.
sync_fails	int	Aantal opeenvolgende keren dat niet alle camServer's dezelfde trip ID mogen hebben.
state_fails	int	Aantal opeenvolgende keren dat niet alle camServer's zich in dezelfde staat mogen bevinden.
sate_mismatch_percentage	double	Percentage van camServer's dat zich in dezelfde staat moet bevinden alvorens andere camServer's automatisch hun eigen staat zullen bijwerken.
server	string	URL van de <i>wayside server</i> waar de resultaten naartoe worden gestuurd.

Tabel 5.4: Lijst configuratieparameters camServer

# Hoofdstuk 6

## Ontwikkeling back-end

### 6.1 Databank

Voor de functionaliteit en uitzicht van de back-end besproken worden, zullen eerst de achterliggende databank worden toegelicht. Alle informatie wordt opgeslagen in drie tabellen in een MySQL - databank op de *wayside server*. De eerste tabel, *railtrips*, bevat een lijst van trajectnummers en hun bijhorende haltes. De tweede tabel, *railrecords*, bevat de tellingsresultaten zoals reeds beschreven in tabel 5.2. De derde tabel, *railreports*, bevat de logs van de APC - infrastructuur zoals reeds werd beschreven in tabel 5.3. De lay-out van *railtrips* is te vinden in tabel 6.3 en is voor verdere tests opgevuld met alle weektrajecten van de Belgische treinoperator NMBS [3]. De lay-out van de tabellen *railreports* en *railrecords* zijn respectievelijk terug te vinden in tabellen 6.2 en 6.1.

### 6.2 Overzicht back-end

In België worden naar schatting 3600 trajecten per dag gereden. Mocht elk traject uitgevoerd worden met een APC - systeem aan boord, dan resulteert dit in 3600 detectieresultaten per dag of 108 000 detectieresultaten per maand [3], wat nog eens moet worden vermenigvuldigd met het aantal haltes per individueel traject. Er is dus nood aan een goed filter- en vergelijkingssysteem om orde in deze grote hoeveelheid data te scheppen.

Kolom	Beschrijving
ID (int)	volgnummer van de rij
TRIPID (text)	uniek trip_id
TRAJECTORY (text)	traject_id of trajectnummer
TIMESTAMP (int)	tijdstip waarop het record werd gegenereerd
STATION (text)	naam van de halte waar de telling plaatsvond
IN (int)	aantal ingestapte passagiers
OUT (int)	aantal uitgestapte passagiers
GATHERED (int)	aantal gebruikte camNode's voor telling
TOTAL (int)	totaal aantal toegewezen camNode's

Tabel 6.1: Lay-out van de MySQL - tabel *railrecords*.

Kolom	Beschrijving
ID (int)	volgnummer van de rij
TRIPID (text)	uniek trip_id
SOURCE (text)	ID van de rapporterende hardware
TARGET (text)	ID van het traject of veroorzakende hardware
ERROR (int)	foutcode, zie tabel 5.3
TIMESTAMP (int)	tijdstip waarop het record werd gegenereerd

Tabel 6.2: Lay-out van de MySQL - tabel *railreports*.

Kolom	Beschrijving
TRAJECT (text)	traject_id of trajectnummer
BEGIN (text)	naam beginhalte
END (text)	naam eindhalte
STOPS (text)	lijst van tussenliggende haltes

Tabel 6.3: Lay-out van de MySQL - tabel *railtrips*.

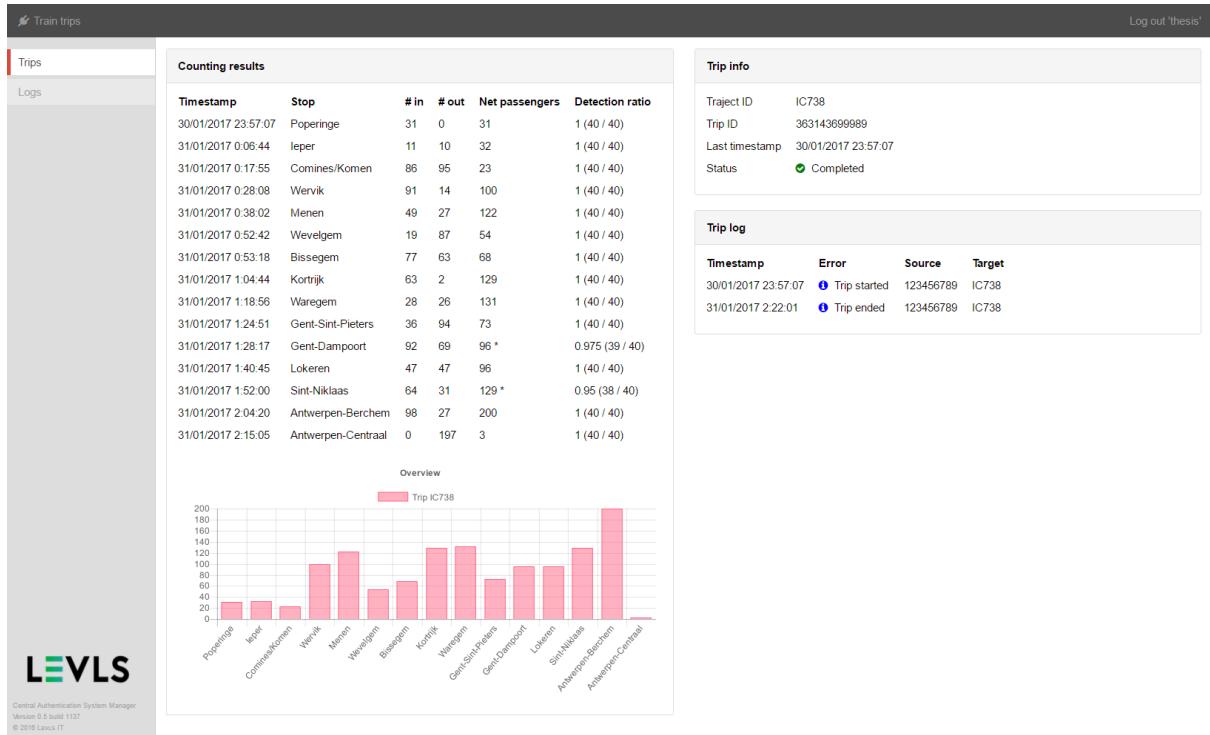
Om de back-end makkelijk toegankelijk te maken werd deze geïmplementeerd in PHP op een publieke server. Als framework werd het closed source framework *LEVLS CAS* gebruikt als hulpmiddel om ontwikkeling te versnellen en om een loginsysteem te voorzien, maar de basis-functionaliteit van de back-end kan makkelijk worden overgezet naar een ander open source framework zoals *PHPixie* [29] of *Symfony* [30], eventueel in combinatie met een loginsysteem zoals *OAuth* [31].

De back-end bevat twee secties. De sectie *trips* bevat een lijst van trips en hun bijhorende status. Vanuit deze pagina is het ook mogelijk om een vergelijkende grafiek te maken van verschillende trajecten of om de huidige lijst te filteren op traject en datum. Tot slot kunnen van hieruit de details van een traject worden weergegeven: de tellingsresultaten en de APC - systeemlogs (zie tabel 5.3). Om de data te visualiseren in een grafiek werd de open source javascript library *chart.js* gebruikt.

De tweede sectie bevat een lijst van APC - systeemlogs, gesorteerd op tijdstip van gebeuren (zoals te zien in figuur ??). Deze lijst kan gefilterd worden op datum en foutniveau (*notices*, *warnings* en *errors*). Aan de hand van deze logs kunnen technici de gezondheid van het APC - systeem en verschillende geregistreerde hardwarecomponenten monitoren. Zoals reeds vermeld in paragrafen 4.3.1 en 5.2.1 heeft elke APC - component (camNode of camServer) een uniek hardware-gerelateerd ID waaraan zaken kunnen worden gekoppeld zoals het identificatienummer van het treinstel waarin de component werd geïnstalleerd.

### 6.3 Visualisatie van een rit

In het visualisatiescherm van één specifieke treinrit (figuur 6.1) kunnen de tellingsresultaten in het blok *Counting results* bekijken worden in tabel- en grafiekvorm. In de tabel staan de verschillende resultaten per station opgelijst. De kolom *Net passengers* geeft aan hoeveel passagiers er in totaal in de trein zaten op het moment dat de trein het station weer verliet en de telling gestopt werd. Het is de info uit deze kolom die in de grafiek gebruikt wordt.



Figuur 6.1: Visualisatie van een treinrit

De kolom *detection ratio* bevat het percentage van camNode's dat gebruikt werd om de detectie uit te voeren. Wanneer er door een storing een camNode verbinding zou verliezen kan het zijn dat de camServer's wel nog doorgaan met detectie (afhankelijk van het uitvalpercentage *camnode\_fail\_percentage* in de camServer configuratie, zie tabel 5.4). De back-end zal de resultaten dan markeren met een sterretje (\*) en de resultaten interpoleren, dit wordt verder uitgelegd in paragraaf 6.5.

In het blok *Trip info* kunnen zaken zoals het trajectnummer, tripnummer, de laatste tijdstempel en de status van het traject weergegeven worden. Een treinrit kan 4 mogelijke statussen hebben, deze zijn opgeliist in tabel 6.4.

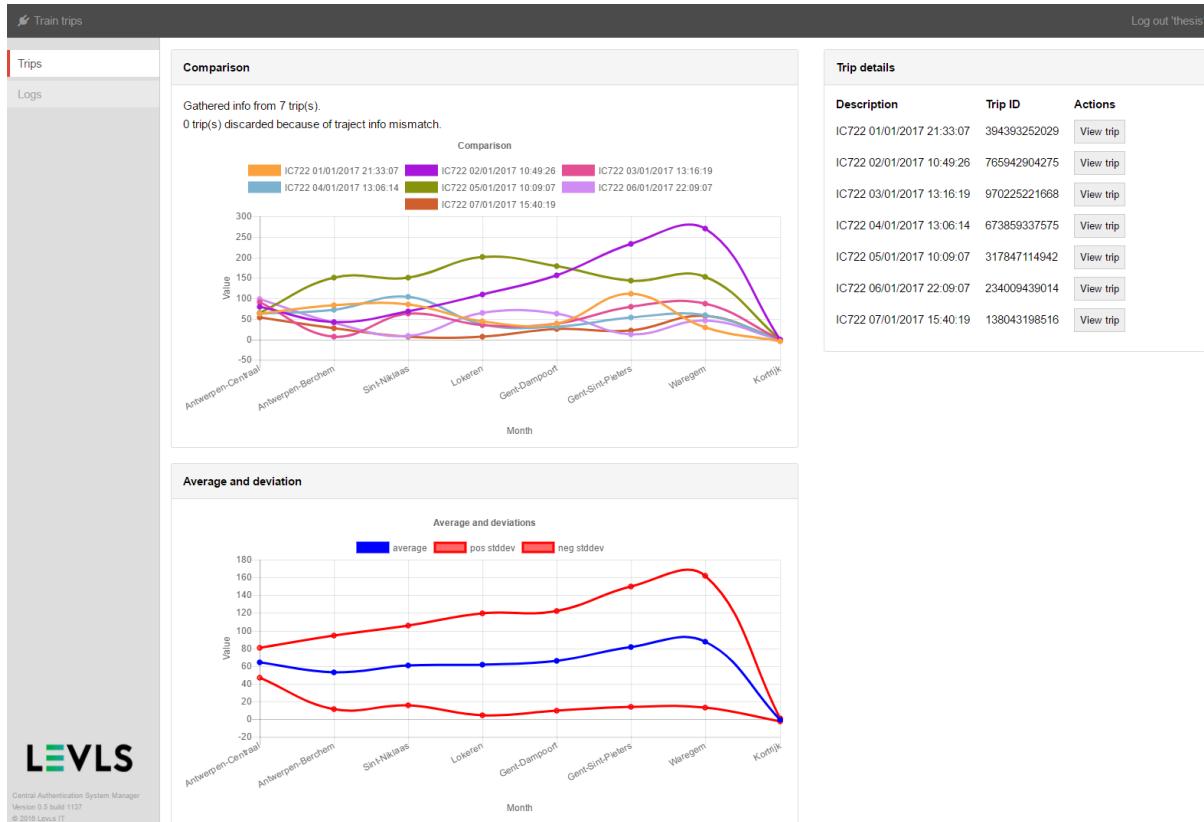
## 6.4 Vergelijking van trajecten

Om makkelijk trajecten met elkaar te vergelijken is er een vergelijkingswizard ontwikkeld met volgende opties:

- *Type*: *Traject ID historical data* of *Full traject historical data*. Het eerste verzamelt alle resultaten met hetzelfde trajectnummer, het tweede zoekt naar alle trajectnummers die dezelfde haltes hebben als het geselecteerde trajectnummer. Een trajectnummer is in dit geval uniek gekoppeld aan een traject op een bepaald uur van de dag, maar niet aan een unieke set van haltes. IC712 en IC714 doorlopen bijvoorbeeld exact hetzelfde traject, maar op andere tijdstippen van de dag [3].
- *Traject ID*: het trajectnummer, te selecteren uit de beschikbare trajecten in de tabel *rail*-

Status	Beschrijving
Completed (with errors)	Voor dit traject werd foutcode 0 (traject gestart) en foutcode 20 (traject afgerond) gerapporteerd. Indien er daarnaast foutcodes van niveau 1 gerapporteerd werden zal het opschrift <i>with errors</i> worden bijgevoegd.
In progress (with errors)	Voor dit traject werd minder dan 12 uur geleden enkel foutcode 0 (traject gestart) gerapporteerd. Indien er daarnaast foutcodes van niveau 1 gerapporteerd werden zal het opschrift <i>with errors</i> worden bijgevoegd. Indien het laatste rapport ouder is dan 12 uur verandert de status naar <i>assumed dead</i> .
Aborted	Voor dit traject werd foutcode 0 (traject gestart) en één of meerdere foutcodes van niveau 2 gerapporteerd, wat betekend dat het APC - systeem de detectie beëindigd heeft wegens een onherstelbare fout.
Assumed dead	Trajecten die als <i>in progress</i> gemarkeerd werden maar langer dan 12 uur geen nieuwe rapportering hebben gedaan krijgen de status <i>assumed dead</i> . Ook hier kan worden aangenomen dat het APC - systeem de detectie heeft beëindigd wegens een onherstelbare fout.

Tabel 6.4: Statuscodes van een specifieke treinrit.



Figuur 6.2: Vergelijking van verschillende treinritten

The screenshot shows a 'New comparison' wizard window with the following settings:

- Type: Traject ID historical data
- Traject ID: IC720
- Timespan: Date range
- From: 01-01-2017, 00:00
- To: 08-01-2017, 00:00
- Generate chart

Figuur 6.3: Vergelijkingswizard

*trips.*

- *Timespan*: *Date range* of *Select dates*. Met het eerste kan een start- en eindmoment worden gekozen waartussen de resultaten moeten liggen. Met het tweede kunnen exacte data worden geselecteerd (bijvoorbeeld alle maandagen van de maand januari).

Na het ingeven van de opties worden alle trajecten gezocht die aan de criteria voldoen en een status *Completed (with errors)* hebben. Vervolgens verschijnt het vergelijkingsscherm uit figuur 6.4. In het blok *Comparison* verschijnt een lijngrafiek met de verschillende telresultaten van de verschillende trajecten, elk in een andere kleur. Elke waarde represeneert het aantal vervoerde passagiers tussen deze halte en de volgende.

In het blok *Average and deviation* wordt het gemiddeld aantal passagiers van alle geselecteerde trajecten weergegeven in het blauw. Om de stabiliteit van de metingen weer te geven wordt er ook een plot gemaakt van dat gemiddelde plus en min de standaarddeviatie, in het rood op dezelfde grafiek.

Tot slot kan in het blok *trip details* doorgeklikt worden naar de detailpagina's van de verschillende afgebeelde treinritten.

## 6.5 Fouttolerantie

Zoals in voorgaande paragrafen reeds werd beschreven houdt de back-end rekening met het aantal camNode's die gebruikt werden tijdens de passagierstelling. Als er tijdens een treinrit enkele camNode's uitvallen en het percentage van het aantal werkende camNode's boven de grens *camnode\_fail\_percentage* ligt zal de detectie gewoon doorgaan. Per halteresultaat zal het aantal gebruikte camNode's ten opzichte van het aantal toegewezen camNode's worden meegestuurd naar de *wayside server*. De back-end zal dan aan de hand van volgende formule de aantallen interpoleren, met *TOTAAL* het totaal aantal toegewezen camNode's en *VERZAMELD* het totaal aantal nog werkende camNode's op moment van detectie.

$$INbijgestuurd = IN \cdot \frac{TOTAAL}{VERZAMELD}$$

$$OUTbijgestuurd = OUT \cdot \frac{TOTaal}{VERZAMELD}$$

Dergelijke geïnterpoleerde resultaten zullen in het *Counting results* blok van het tripvenster aangeduid worden met een sterretje (\*) in de kolom *Net passengers*.

Het uitvallen van camNode's of detectiefouten kunnen ervoor zorgen dat de resultaten afwijken van de werkelijkheid. Soms kunnen de aantallen negatief worden, bijvoorbeeld als de passagiers ten onrechte als 'naar buiten gewandeld' werden gemarkerd of als instappende passagiers niet konden worden gedetecteerd. Wanneer dit gebeurt zal het passagiersaantal op die stop automatisch als 0 worden weergegeven en worden aangeduid met een dubbel sterretje (\*\*).

The screenshot shows a user interface for viewing system logs. At the top, there is a header bar with a magnifying glass icon and the text 'Filter logs'. Below this is a section titled 'Logs and reports' with a sub-section 'Previous 50 records'. The main area contains a table with the following data:

	Timestamp	Trip ID	Generated by	Caused by	Error type	Actions
●	22/05/2017 13:36:56	1670725321	68676	IC811	Trip ended (20)	<button>Open trip</button>
●	22/05/2017 13:29:00	1670725321	68676	IC811	Trip started (0)	<button>Open trip</button>
●	22/05/2017 13:27:52	3134866926	68676	field-test-2	Trip ended (20)	<button>Open trip</button>
●	22/05/2017 13:25:34	3134866926	68676	field-test-2	Trip started (0)	<button>Open trip</button>
●	22/05/2017 13:24:08	3884438901	80478	field-test-1	Trip started (0)	<button>Open trip</button>
●	22/05/2017 13:19:21	645697330	74154301	110631	Camserver lost (2)	<button>Open trip</button>
●	22/05/2017 13:19:21	645697330	74154301	n/a	Force terminated (5)	<button>Open trip</button>
●	22/05/2017 13:18:33	645697330	93017	field-test-v8	Trip started (0)	<button>Open trip</button>
●	22/05/2017 13:17:55	3470013961	93017	110631	Camserver lost (2)	<button>Open trip</button>
●	22/05/2017 13:17:55	3470013961	93017	n/a	Force terminated (5)	<button>Open trip</button>
●	22/05/2017 13:17:08	3470013961	93017	field-test	Trip started (0)	<button>Open trip</button>
⚠	22/05/2017 13:17:06	3470013961	74154301	3477687274	Camnode lost (1)	<button>Open trip</button>

Figuur 6.4: Logscherm met alle APC - systeemlogs

# Hoofdstuk 7

## Evaluatie

In dit hoofdstuk worden de verschillende schaalbaarheids- en performantietests van camServer, camNode, de back-end en het algoritme voor passagiersdetectie besproken.

### 7.1 Evaluatietools

Om een simulatie van een echte treinrit te kunnen uitvoeren zijn twee extra tools ontwikkeld. Als eerste is er een *trainframe emulator* dat de taken van de boordsystemen op de trein moet simuleren. Het is de taak van deze tool om aan camServer te signaliseren wanneer een traject geïnitialiseerd wordt, wanneer en waar de trein stopt, wanneer de trein weer vertrekt en wanneer het traject beëindigd wordt. Deze tool is tevens in staat om commando's naar camNode's te sturen.

Daarnaast is er een afgeslankte versie van camNode ontwikkeld die een vooraf opgenomen videobestand in een lus kan doorsturen in plaats van een rechtstreekse camerafeed. Het identificatienummer van deze testnode wordt willekeurig gekozen bij het starten van de applicatie en is niet hardwareafhankelijk, in tegenstelling tot de originele camNode implementatie. Dit maakt het mogelijk om verschillende testnodes uit te voeren op dezelfde computer zonder dat er een ZeroConf aankondigingsconflict optreedt. Tot slot is de testnode ook in staat de hele video in het RAM - geheugen te laden zodat er geen bottleneck optreedt wanneer verschillende instanties tegelijkertijd van de harde schijf proberen lezen.

### 7.2 Schaalbaarheid camServer en camNode

Om de schaalbaarheid van camServer en camNode te testen werd een simulatie gedaan van één van de grootste treinen uit de NMBS - vloot. IC717 van Antwerpen-Centraal naar Poperinge telt 9 passagierswagens met elk 2 ingangen per kant [3]. Dit resulteert in 36 te controleren ingangen.

Met uitbreidingsmogelijkheden in het achterhoofd werden de tests uitgevoerd met 40 testnodes en respectievelijk 1, 2 en 3 camServer's. Alle instanties werden verdeeld over 6 computers waarvan de hardwarespecificaties terug te vinden zijn in tabel 7.1. Hierbij werden bandbreedte-, processor- en geheugengebruik opgemeten. Om deze verbruiken te meten werd gebruik gemaakt van de ingebouwde *System Monitor* in *Ubuntu 16.04 LTS*.

```

lorre851@smartfix-ubuntu: ~/CLionProjects/trainframe_emu/cmake-build-debug
[2] 68676 @ smartfix-ubuntu.local on tcp://192.168.0.103:5782
-- end of list
switch
switched to camnode broadcast
list
-- camNode list --
[0] 1135521842 @ raspberrypi.local on tcp://192.168.0.137:5008
[1] 1151675447 @ zotac-desktop.local on tcp://192.168.0.169:6020
[2] 1226329593 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5010
[3] 1352259133 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5040
[4] 1434728213 @ zotac-desktop.local on tcp://192.168.0.169:5000
[5] 1547842639 @ zotac-desktop.local on tcp://192.168.0.169:5900
[6] 1591458442 @ zotac-desktop.local on tcp://192.168.0.169:5902
[7] 1673502991 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5006
[8] 1683269378 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5022
[9] 1684060399 @ zotac-desktop.local on tcp://192.168.0.169:5550
[10] 172659029 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5000
[11] 1737337004 @ raspberrypi.local on tcp://192.168.0.137:5020
[12] 1807187910 @ zotac-desktop.local on tcp://192.168.0.169:5100
[13] 1863516123 @ smartfix-dev-hp.local on tcp://192.168.0.248:5010
[14] 1954141398 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5060
[15] 1958078165 @ zotac-desktop.local on tcp://192.168.0.169:5560
[16] 1986074647 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5018
[17] 2020193871 @ raspberrypi.local on tcp://192.168.0.137:5000
[18] 2303540494 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5004
[19] 2369545059 @ raspberrypi.local on tcp://192.168.0.137:5004
[20] 2572797625 @ zotac-desktop.local on tcp://192.168.0.169:6022
[21] 2625121335 @ zotac-desktop.local on tcp://192.168.0.169:5040
[22] 2817629791 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5008
[23] 3136854841 @ zotac-desktop.local on tcp://192.168.0.169:5904
[24] 3141121322 @ zotac-desktop.local on tcp://192.168.0.169:5020
[25] 3427456405 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5012
[26] 3477687274 @ raspberrypi.local on tcp://192.168.0.137:5002
[27] 3541288480 @ zotac-desktop.local on tcp://192.168.0.169:5555
[28] 379613771 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5020
[29] 3809927600 @ raspberrypi.local on tcp://192.168.0.137:5006
[30] 395001025 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5044
[31] 4119497433 @ smartfix-dev-hp.local on tcp://192.168.0.248:5004
[32] 4263435427 @ zotac-desktop.local on tcp://192.168.0.169:6026
[33] 479708831 @ zotac-desktop.local on tcp://192.168.0.169:5580
[34] 57736904 @ smartfix-dev-hp.local on tcp://192.168.0.248:5002
[35] 593384439 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5016
[36] 617917051 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5080
[37] 632120518 @ smartfix-ubuntu-dev.local on tcp://192.168.0.99:5024
[38] 700449564 @ smartfix-dev-hp.local on tcp://192.168.0.248:5000
[39] 81037893 @ zotac-desktop.local on tcp://192.168.0.169:6024
-- end of list
switch
switched to camserver broadcast
list
-- camServer list --
[0] 74154301 @ smartfix-ubuntu.local on tcp://192.168.0.103:10641
[1] 80478 @ smartfix-ubuntu.local on tcp://192.168.0.103:5090
[2] 68676 @ smartfix-ubuntu.local on tcp://192.168.0.103:5782
-- end of list
init field-test-2
[0] recv: ack
[1] recv: ack
[2] recv: ack

```

Figuur 7.1: *Trainframe emulator* met een lijst van de beschikbare camNode's en camServer's in het netwerk

Naam	Netwerksnelheid	CPU	Geheugen	Bezetting
zotac-desktop	1 Gbps full duplex	Intel Atom D510 @ 1.6 GHz dual core	4 GB DDR2	15 testnodes
smartfix-ubuntu-dev	1 Gbps full duplex	AMD A4-3300M @ 1.9 GHz dual core	6 GB DDR2	16 testnodes
smartfix-ubuntu	1 Gbps full duplex	Intel Core i7-3610QM @ 2.3 GHz octa core	16 GB DDR3	1 of 2 cam-Server
smartfix-dev-hp	100 Mbps full duplex	Intel Core i3-2330M @ 2.2 GHz quad core	6 GB DDR3	3 testnodes
raspberrypi	1 Gbps full duplex	ARMv8 @ 1.2 GHz quad core	1 GB DDR2	6 testnodes
odroid	1 Gbps full duplex	Samsung Exynos5422 Cortex A15 @ 2Ghz octacore	2 GB DDR3	0 of 1 cam-Server

Tabel 7.1: Lijst van testhardware voor evaluatie.

### 7.2.1 Bandbreedte

Voor de volledige test werd gestart werd het netwerkverbruik van één enkele camNode opgemeten. Wanneer de camNode tussen de 10 en 15 frames per seconde doorstuurt op een resolutie van 640x360 pixels met 1 byte per pixel ligt het verbruik gemiddeld op 2,8 MB/s.

Vervolgens werd een test uitgevoerd waarbij 40 testnodes op 4 verschillende computers werden gestart, waarna één camServer op een afzonderlijke computer draaide. Hier steeg het netwerkverbruik naar gemiddeld 100 MB/s. De afwijking op de theoretische 112 MB/s (40 keer 2,8) is te wijten aan het feit dat niet elke testnode op zijn maximale framerate kon draaien door lichte overbelasting van de *zotac-desktop* testcomputer. Beide cijfers komen echter dicht in de buurt van de theoretische maximumsnelheid van een gigabitnetwerk (125 MB/s, omgerekend 1000 Mb/s), dergelijke belasting is niet wenselijk.

Wanneer er twee camServer's op afzonderlijke computers worden uitgevoerd zakt het netwerkverbruik naar gemiddeld 50 MB/s per camServer. Bij gebruik van 3 camServer's werden gemiddelen van 35 MB/s opgemeten. De conclusie die hieruit kan worden gemaakt is dat de netwerkdruk kan worden beperkt als de videostreams over verschillende camServer - machines worden verdeeld. Een netwerk met snelheden van minstens 1 Gbps is echter wel vereist. Om de netwerkdruk verder te reduceren is implementatie van videostream aan variabele bitrate en bijkomende beeldcompressie interessant voor verder onderzoek.

Bovengenoemde waarden zijn enkel op te meten wanneer detectie actief is (tussen het ontvangen van een *halt* en *depart* - commando van de *trainside*). Wanneer de detectie niet actief is, is het bandbreedteverbruik verwaarloosbaar (enkele KB/s door heartbeatcommando's).

Process Name	User	% CPU	ID	Memory
camServer	lorre851	13	12624	70,9 MiB
camServer	lorre851	10	12285	106,9 MiB
camServer	lorre851	9	12271	87,8 MiB

Figuur 7.2: Geheugen- en processorverbruik camServer in Ubuntu System Monitor

### 7.2.2 Geheugen- en processorgebruik

In dezelfde testopstelling werd ook het geheugenverbruik gemeten. Hierbij werden drie camServer's uitgevoerd, twee met 13 en één met 14 camNode's. De *master* camServer had een geheugenverbruik van 106,9 MB met 13 toegewezen camNode's. De overige twee camServer's hadden met 13 en 14 camNode's een geheugenverbruik van respectievelijk 70,9 MB en 87,8 MB. Deze cijfers zijn vooral te wijten aan de verschillende buffers die in het algoritme voor passagiersdetectie gebruikt worden. Bovendien worden de bewerkingen steeds op ongecomprimeerde afbeeldingen in het geheugen uitgevoerd. Het verhoogde geheugengebruik van de *master* is te wijten aan de extra gegevens die deze camServer moet bijhouden (tellingsresultaten en lijsten van alle camNode's en camServer's in het netwerk).

Opnieuw in dezelfde testopstelling werd het processorverbruik opgemeten met de *Ubuntu System Monitor*. Wanneer de camServer zich in de staat *Detecting* bevindt schommelt het processorgebruik tussen de 5 en 15 procent op een *Core i7-3610QM*. Het verschil tussen *master* en *niet-master* camServer's is niet merkbaar. Wanneer de camServer zich in een andere staat bevindt is het processorgebruik verwaarloosbaar (0 tot 1 procent).

## 7.3 Schaalbaarheid back-end

Om de schaalbaarheid van de back-end te testen werd de database eerst gevuld met testdata. Hiervoor werden 3376 trajecten gebruikt van de Belgische treinoperator NMBS [3], elk met hun bijhorende tussenliggende stops. Voor elk traject werden telrecords gegenereerd voor elke halte met willekeurige aantallen in- en uitgestapte passagiers. Hiervoor werden enkele regels toegepast:

- Het aantal uitgestapte passagiers in een beginhalte is altijd gelijk aan nul
- Het aantal ingestapte passagiers in een eindhalte is altijd gelijk aan nul
- Het aantal uitgestapte passagiers in een halte kan nooit groter zijn dan het aantal passagiers dat op dat moment werden vervoerd

Dit proces werd voor elk traject 31 keer herhaald (1 keer per dag), wat een totaal van 1 065 035 tellingsresultaten opleverde. Daarnaast werd in 10 procent van de gevallen een willekeurig aantal camNode's als 'defect' gemarkeerd. Dit willekeurig aantal lag tussen de 0 en 10 procent van het totaal aantal toegewezen camNode's. Vervolgens werd de tijd opgemeten die nodig was om enkele basiszaken uit te voeren zoals het toepassen van zoekfilters en het vergelijken van kleine en grotereritten. Deze tijden zijn terug te vinden in tabel 7.2.

De meest voor de hand liggende vergelijkingen hebben tussen de 5 en de 15 seconden nodig om te laden. Dit omvat zaken zoals vergelijkingen van trajectnummers over een tijdspannen van twee weken, vergelijking van equivalenten trajecten doorheen één dag of vergelijking van een

Opdracht	Tijdspannes	Haltes	Ritten	Laadtijd
Filter IC722	01/01/2017 t.e.m. 14/01/2017	8	n.v.t.	3 sec
Filter IC730 + equivalent	01/01/2017 t.e.m. 14/01/2017	15	n.v.t.	7 sec
Filter IC728 + equivalent	01/01/2017 t.e.m. 14/01/2017	15	n.v.t.	7 sec
Filter IC908	5 maandagen van januari 2017	7	n.v.t.	1 sec
Filter IC535 + equivalent	5 maandagen van januari 2017	11	n.v.t.	15 sec
Vergelijking IC704	01/01/2017 t.e.m. 14/01/2017	15	13	10 sec
Vergelijking IC728 + equivalent	01/01/2017 t.e.m. 14/01/2017	15	262	163 sec
Vergelijking IC908	5 maandagen van januari 2017	7	5	5 sec
Vergelijking IC535 + equivalent	5 maandagen van januari 2017	11	81	74 sec
Vergelijking IC3319 + equivalent	01/01/2017	14	14	13 sec

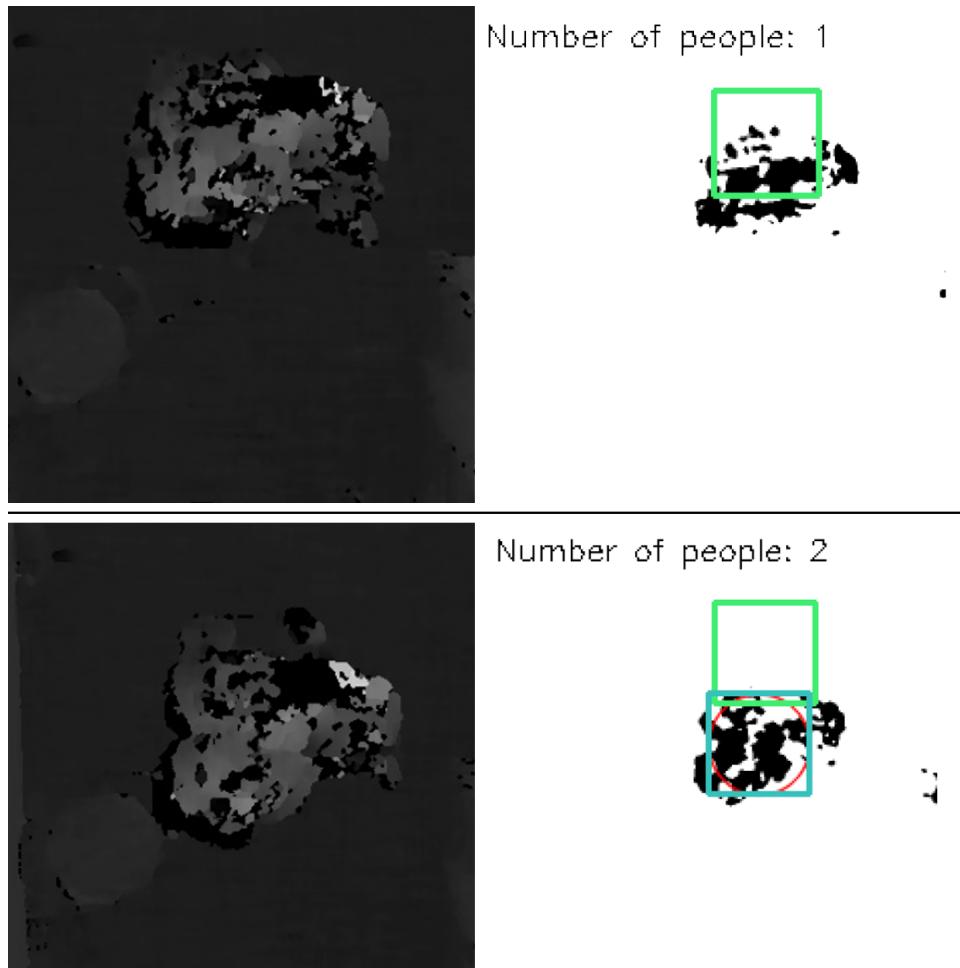
Tabel 7.2: Laadtijden voor vergelijkingen en filters in de back-end

trajectnummer op elke eerste dag van de week. Telkens werden tussen de 5 en de 15 ritten verzameld. Wanneer het aantal ritten groter wordt dan 40 stijgt de rekentijd drastisch (meer dan 40 seconden) en wordt de resulterende grafiek zo goed als onbruikbaar door de grote hoeveelheid datapunten. Dit laatste scenario treedt vooral op wanneer equivalenten trajecten over verschillende data worden geplot. De back-end is voor dergelijke vergelijkingen dus ongeschikt te noemen. Bovendien zijn vergelijkingen van individuele trajectnummers over een bepaalde tijdspanne of equivalenten trajecten binnen één dag interessanter dan dergelijke vergelijkingen van equivalenten trajecten over een grote tijdspanne.

Het toepassen van zoekfilters is echter geen probleem en neemt tussen de 8 en 15 seconden in beslag, zelfs bij meer ingewikkelde zoekopdrachten met equivalenten trajecten over tijdspannes.

## 7.4 Microcomputers

De camNode software werd getest op drie verschillende apparaten: een *Raspberry Pi 3 model B*, een *Odroid XU4* en een *Zotac ZBOX*. Hun hardwarespecificaties werden reeds opgeliist in tabel 7.1. De dieptebeelden werden gegenereerd met twee *Logitech C525* webcamera's op een resolutie van 640x360 pixels. Al snel bleek het *semiglobal blockmatching* algoritme, dat in de tests op een Intel Core i7 nog het beste presteerde ten opzichte van andere OpenCV - algoritmes, te zwaar was voor zowel de Raspberry Pi als de Odroid. De Raspberry Pi haalde slechts framerates van 0,4 frames per seconde, de Odroid haalde slechts 4 frames per seconde. Dit is te weinig om de *tracking* van het algoritme voor passagiersdetectie correct te laten werken. Beide apparaten slaagden er wel in om een gewone RGB - videofeed door te sturen aan respectievelijk 15 en 25



Figuur 7.3: Matching- en trackingfouten ten gevolge van imperfecte dieptemap

frames per seconde. Hieruit kan worden besloten dat het aansturen van een gespecialiseerde dieptecamera wel mogelijk zou zijn met één van deze microcomputers. Het SGBM - algoritme is echter te zwaar voor beide ARM - computers om vlot te kunnen werken.

Tot slot werd camNode uitgevoerd op een Zotac ZBOX met x86 processorarchitectuur. Deze test faalde echter vermits dit toestel slechts over één USB - bus beschikt waardoor er slechts één USB - camera tegelijk kan worden aangesproken. Een mogelijke oplossing hiervoor is het opnieuw compileren van OpenCV met MJPEG - ondersteuning in plaats van FFMPEG - ondersteuning. Dit moet de benodigde bandbreedte per webcam reduceren waardoor er meerdere camera's via één USB - bus kunnen worden aangestuurd. De webcam moet echter MJPEG - encoding ondersteunen opdat deze aanpak zou werken.

## 7.5 Stereovisie en persoonsdetectie

Het SBGM - algoritme waarmee dieptebeelden worden gegenereerd vertoont imperfecties. Zo als reeds werd uitgelegd in paragraaf 4.2.2 komt het soms voor dat regio's in de afbeeldingen niet kunnen worden gematcht, wat resulteert in zwarte regio's in de dieptebeelden. Dit zorgt op zijn beurt voor problemen in het algoritme voor blobdetectie. Door de imperfecte matches

kunnen de blobs versnipperd geraken door deze zwarte storing (zoals geïllustreerd in figuur 7.3), wat kan resulteren in onterechte detectie van meerdere personen (in geval van versnippering in grote afzonderlijke blobs) of het niet detecteren van een persoon (in geval van versnippering in blobs die door hun te kleine oppervlakte worden genegeerd). Deze incorrecte matching is het gevolg van de vertraging tussen beide camera's bij het binnennemen van frames, het feit dat beide camera's beschikken over een oncontroleerbare autofocus en de imperfectie van het SGBM - algoritme zelf. Daarnaast zorgt de variabele rekentijd van het SGBM - algoritme soms voor het wegvalLEN van frames. Dit alles zorgt ervoor dat tracking niet altijd mogelijk is en personen niet worden geregistreerd als in- of uitgestapt.

De imperfectie van het algoritme voor stereovisie maakt het met de huidige apparatuur niet mogelijk om het algoritme voor persoonsdetectie op een betekenisvolle manier te evalueren. Perfectioneren van het detectiealgoritme en tests met een gespecialiseerde dieptecamera behoren tot uitbreidingsmogelijkheden en verder onderzoek.

## Hoofdstuk 8

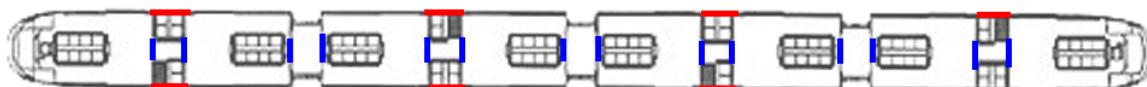
# Uitbreidingsmogelijkheden en verder onderzoek

### 8.1 Tellingen in wagons

Het APC - systeem zou kunnen worden ingezet om tellingen in individuele wagons te kunnen uitvoeren. Zo kunnen aan de buitenkant van de wagons schermen met tellers worden bevestigd zodat passagiers op het perron een beter zicht hebben waar ze best kunnen instappen.

Om dit te bereiken zouden de camNode's in twee groepen moeten gesplitst: een *uitgangsgroep*, die zijn normale functionaliteit behoudt en het globaal aantal in- en uitgestapte passagiers telt, en een *doorgangsgroep*, die de tellingen in de wagons zelf zal uitvoeren. De camera's van de doorgangsgroep zullen aan elke doorgang van een wagon worden gemonteerd. Daarnaast zal elke wagon een uniek identificatienummer moeten krijgen zodat elke camNode weet in welke wagon deze werd gemonteerd.

Ook hier zal de detectie door de verschillende camServer's gebeuren. Het verschil is dat de tellingsresultaten periodiek moeten kunnen worden opgehaald terwijl de detectie nog aan de gang is. Daarnaast zou de detectie ook gedurende het hele traject moeten worden uitgevoerd en is dus onafhankelijk van *halt* of *depart* commando's. Het aantal passagiers in een wagon is dan gelijk aan het verschil tussen het totaal aantal ingestapte en uitgestapte passagiers in die wagon. Het is de taak van de *master camServer* om periodiek de resultaten van de *doorgangsgroep* op te halen, het netto aantal passagiers per wagon te berekenen en de tellers op de wagons bij te werken. Verder moet de *watchdog* worden aangepast zodat het wegvalLEN van te veel camNode's uit de *doorgangsgroep* niet zou leiden tot een volledige reset van het systeem. Na het uitvoeren



Figuur 8.1: Schematische voorstelling van de posities van de uitgangsgroep (rood) en doorgangsgroep (blauw)

van een *terminate* commando zouden de tellers dan weer gereset worden.

## 8.2 Doorrijdende treinen met gewijzigd traject / samenstelling

Het komt soms voor dat treinstellen aan hun eindhalte doorrijden onder een ander trajectnummer. Wanneer er passagiers op de trein blijven zitten tijdens de trajectwissel zal dit in de huidige opstelling leiden tot problemen met negatieve passagiersaantallen. Om dit te kunnen opvangen zou een afzonderlijk commando moeten worden geïmplementeerd dat in staat stelt de eindresultaten van de telling in het vorige traject op te slaan en opnieuw te gebruiken in het nieuwe traject. In dergelijke situatie is het wenselijk om drie nieuwe soorten beëindigingscommando (zoals *terminate*) te implementeren voor de drie mogelijke scenario's die zich kunnen voordoen:

- Een treinstel rijdt met dezelfde wagons door onder een ander trajectnummer.
- Een treinstel wordt aan een ander treinstel gekoppeld en rijdt door onder een ander trajectnummer.
- Een treinstel wordt gesplitst waarna beide segmenten doorrijden onder een ander trajectnummer of naar de stelplaats gaan.

In het eerste scenario is het wenselijk om een 'halve reset' uit te voeren: de verdeling van camNode's en camServer's blijft dezelfde, de telresultaten worden doorgestuurd naar de *wayside server* en de tellers worden gereset, maar het netto aantal aanwezige passagiers op het moment van de reset wordt bijgehouden door de *master camServer* en toegevoegd aan het aantal ingestapte passagiers in de eerste halte van het nieuwe traject.

Wanneer er bij een trajectwisseling extra treinstellen worden aangekoppeld én passagiers in de treinstellen blijven zitten zal nog een aanvullend beëindigingscommando moeten worden geïmplementeerd. Bij aankoppeling van het nieuwe treinstel zullen er twee *master camServer*'s in het netwerk te vinden zijn. Wanneer het *terminate* commando gegeven wordt zullen de camServer's het netto aantal passagiers met elkaar moeten communiceren. Een van de twee camServer's zal dit aantal opslaan en verdergaan als *master* voor het volgende traject. De andere camServer zal zijn status als *master* verliezen. De verdeling van camNode's kan behouden blijven, de camServer's moeten enkel een nieuwe discoveryprocedure starten om de nieuwe camServer's in het netwerk te vinden.

Wanneer er bij een trajectwisseling treinstellen worden gesplitst én passagiers blijven zitten zal een systeem van wagontellingen moeten worden geïnstalleerd om tot een correcte telling te komen. Het APC - systeem heeft in zijn huidige vorm namelijk geen besef van het aantal passagiers in een bepaalde wagon. Voor de afkoppeling geïnitialiseerd wordt zal een extra *master* moeten worden aangeduid zodat er één *master* per segment aanwezig is. Deze keuze kan worden gemaakt als de *trainside software* kan doorgeven welke wagonnummers er deel zullen uitmaken van welk segment. Vervolgens kunnen de *masters* het aantal passagiers per wagon berekenen waaruit het netto aantal passagiers per segment kan worden gebruikt voor de nieuwe telling. Na afkoppeling zullen de overige camServer's gereset moeten worden en zal een nieuw discoveryproces van camNode's en camServer's moeten worden uitgevoerd.

### 8.3 Videostream met variabele bitrate

Om de netwerkdruk te verlagen is het noodzakelijk om beeldcompressie en videostreams met variabele bitrate te implementeren, zeker wanneer een wagontellingssysteem zoals in paragraaf 8.1 zou worden geïmplementeerd. OpenCV bevat reeds een encoderings- en decoderingsfunctie om afbeeldingen te comprimeren met het JPEG 2000 algoritme. Compressie kan als volgt worden geïmplementeerd:

```
1 //output met gecomprimeerde data
2 std::vector<uchar> buffer;
3 //parameters voor compressor
4 std::vector<int> parameters(2);
5 //geef aan dat het om JPEG compressie gaat
6 parameters[0] = cv::IMWRITE_JPEG_QUALITY;
7 //geef de compressieratio op (100 = geen compressie, 0 = maximale compressie)
8 parameters[1] = 80;
9 //encodeer de afbeelding in een buffer
10 cv::imencode(".jpg", mat, buff, param);
```

Om tegemoet te komen aan de variabele bitrate moet bij het doorsturen van elk frame eerst worden geadverteerd hoeveel bytes er zullen worden doorgestuurd. Dit kan gebeuren door een codewoord met vaste grootte door te sturen alvorens de gecomprimeerde afbeelding door te sturen. Dit codeword bevat dan de grootte van de gegevens die zullen volgen, maar kan bijvoorbeeld ook zaken bevatten zoals resolutie en aantal kleurkanalen om zo videostreams op andere resoluties en in kleur mogelijk te maken. In een codeword van 8 bytes zou bijvoorbeeld volgende informatie kunnen worden opgeslagen:

- Resolutie in X - richting (2 bytes, maximum 65 535)
- Resolutie in Y - richting (2 bytes, maximum 65 535)
- Aantal kanalen (1 byte, maximum 255)
- Aantal door te sturen bytes (3 bytes, maximum 16 777 215)

### 8.4 CCTV

Het is met de huidige implementatie van camNode mogelijk om een videofeed van een standaard beeldcamera door te sturen naar een camServer met behulp van het *cam* commando. Dit maakt het mogelijk om het APC - systeem als *closed-circuit television* of surveillancesysteem in te schakelen. Dit maakt het mogelijk om passagiers te filmen bij het in- en uitstappen. Daarnaast kunnen de beelden voor een bepaalde tijd lokaal op de trein worden opgeslagen of naar een *wayside server* gestuurd worden. In ieder geval is beeldcompressie een noodzaak om tegemoet te komen aan de beperkte opslagcapaciteit van de *traininside servers* en beperkte bandbreedte naar de buitenwereld toe.

### 8.5 Opvangen mechanische defecten treinstel

Met het APC - systeem is het mogelijk om mechanische defecten aan deuren van treinstellen te detecteren. Wanneer er herhaaldelijk geen passagiers gedetecteerd worden aan een bepaalde

deur kan de *master camServer* beslissen een rapport door te sturen naar de *wayside server* met de melding dat de deur aan de ingang van de camNode mogelijk defect is.

## 8.6 Bepaling treinstellen over en onder capaciteit

Om een schatting te kunnen maken van de efficiëntie van een traject is er nood aan informatie van de samenstelling van de treinen en de maximale capaciteit van de verschillende wagons. Wanneer per tripnummer een lijst van de gebruikte wagons kan worden doorgestuurd naar de *wayside server* kan de back-end het globale bezettingspercentage per halte van het traject berekenen en zodoende waarschuwingen genereren wanneer een percentage over of onder een bepaalde grens komt te liggen.

Idealiter worden de tellingsresultaten dagelijks buiten de diensturen geëvalueerd door berekening van het bezettingspercentage. Deze evaluatie kan gebeuren met een afzonderlijk PHP - script dat als zogenaamde *cron job* dagelijks op de server wordt uitgevoerd. Trajecten met een te laag of te hoog bezettingspercentage kunnen vervolgens worden opgeslagen in een afzonderlijke lijst voor verdere evaluatie door een administrator.

## 8.7 Herstel na stroomuitval of crash

Om tegemoet te komen aan een plotselinge stroomuitval of crash van een *traininside server* kan een *context save* mechanisme geïmplementeerd worden. Een camServer kan in theorie zonder problemen weer verder werken als bepaalde variabelen uit de klasse *camserver* kunnen worden hersteld vanuit een bestand. Deze variabelen zijn terug te vinden in tabel 8.1. Na de initialisatie van de camServer's moeten de variabelen dan worden weggeschreven naar het bestand. De *master camServer* houdt in datzelfde bestand ook de tot dan toe verzamelde tellingsresultaten bij. Wanneer een traject succesvol wordt beëindigd zal het contextbestand worden verwijderd. Wanneer de staat van de camServer verandert (zie figuur 5.2) moet het contextbestand worden bijgewerkt.

Wanneer het systeem herstart na stroomuitval of crash en een contextbestand in de werkmap terugvindt, dan kunnen de benodigde variabelen opnieuw in het geheugen worden geladen. Het enige dat opnieuw zal moeten gebeuren zijn IP adresresoluties van de verschillende hostnamen van camServer's en camNode's.

## 8.8 Uitbreiding configuratie camNode

Tot slot is het interessant om enkele zaken aan de configuratie van een camNode toe te voegen, zoals de locatie van de camNode in de wagon (linker- of rechterkant, aan een uitgang of in een doorgang van een wagon) en de verschillende thresholds voor het passagiersdetectiealgoritme (beschreven in paragraaf 5.3.2). Deze zaken kunnen makkelijk worden ondergebracht in het configuratiebestand en vergen nog een kleine uitbreiding aan het TCP - commandosysteem om deze metadata op te vragen of te wijzigen.

De locatie van de camNode stelt de camServer in staat om de detectie zo efficiënt mogelijk

Variabele	Beschrijving en opmerkingen
int port	camServer poort van de TCP - interface (wordt normaliter willekeurig gegenereerd bij het opstarten van camServer).
vector<iface_node> nodes	Interfaceobjecten eigen camNode's. Deze objecten vereisen hostnaam, poort en volledige ZeroConf ID als parameters.
string state	Huidige staat van de camServer.
string IC	Nummer van het huidige traject (trajectnummer).
string current_stop	Naam van de huidige halte (indien van toepassing).
int trip_id	Uniek ID van de huidige telling op het traject.
vector<iface_server>	Interfaceobjecten camServer's. Deze objecten vereisen hostnaam, poort en volledige ZeroConf ID als parameters.
bool run_master	Vlag om aan te geven of de camServer in <i>master mode</i> wordt uitgevoerd.
vector<vector<string>> queries	Lijst van tellingsresultaten (enkel beschikbaar op de <i>master camServer</i> ).

Tabel 8.1: Variabelen in *camServer* met essentiële tripinformatie.

uit te voeren. Het is vaak zo dat de trein slechts de deuren opent aan één kant vermits er maar aan één kant een perron aanwezig is. Het is in die gevallen dan ook slechts nodig om detectie in te schakelen voor alle camNode's aan die kant. Momenteel wordt detectie op alle camNode's ingeschakeld, ongeacht of de deuren aan die kant effectief geopend zijn of niet.

Het opslaan van de verschillende thresholds maakt het mogelijk om verschillende soorten dieptecamera's aan de verschillende camNode's te koppelen. Elke dieptecamera heeft zijn eigen features en parameters (zoals de detectierange) wat leidt tot verschillende thresholds om de detectie succesvol te laten werken.

# Hoofdstuk 9

## Conclusie

Na het onderzoeken van de basisaspecten en vereisten van een APC - systeem voor treinen werden drie APC - componenten gedefinieerd: een component (camNode) die detectiesensoren aanstuurt, een component (camServer) die data van de sensoren vergaart en tellingsresultaten doorstuurt naar een server buiten de trein en een component (back-end) die de resultaten verwerkt en visualiseert. Hun interacties met elkaar en met de bestaande infrastructuur werden besproken. Vervolgens werd de ontwikkeling van deze drie componenten in detail beschreven. Daarna werden de componenten geëvalueerd op schaalbaarheid en performantie.

Uit de tests met camNode bleek dat de twee gebruikte microcomputers, de Raspberry Pi 3 model B en de Odroid XU4, niet sterk genoeg waren om dieptebeelden te genereren vanaf twee standaard beeldcamera's. Zij haalden respectievelijke snelheden van 0,4 en 4 frames per seconde. Bij het doorsturen van één standaard videofeed werd respectievelijk 15 en 25 frames per seconde behaald. De conclusie hieruit is dat deze microcomputers geschikt zijn voor het aansturen van één gespecialiseerde stereografische camera, maar niet voor het genereren van dieptebeelden vanaf twee standaardcamera's.

De evaluatie van de camServer software bestond uit schaalbaarheids- en performantietests. Wanneer de software gedraaid werd op een computer met een Intel Core i7-3610QM processor waren er geen problemen op te merken. Tijdens passagiersdetectie gebruikte het programma gemiddeld 70 tot 110 MB geheugen met een processorbelasting van 5 tot 15 procent. Het doorsturen van videotstreams vormde echter een zware belasting voor het netwerk: wanneer er 40 camNode's naar 1 camServer streamen ligt het netwerkverbruik rond de 100 MB/s. Wanneer de camNode's echter verdeeld worden over 2 of 3 camServer's zakt het verbruik naar respectievelijk 50 MB/s en 35 MB/s. Hieruit kan worden besloten dat distributie van de camNode's over verschillende camServer's een goede techniek is om de netwerkdruk te verlichten. Implementatie van beeldcompressie blijft echter wel een interessante piste voor uitbreiding en verder onderzoek om het netwerk meer te onlasten.

Om het algoritme voor passagiersdetectie op een betekenisvolle manier te evalueren waren onvoldoende middelen beschikbaar. Het algoritme voor stereovisie maakt te veel fouten die er te vaak voor zorgen dat de passagiersdetectie- of tracking faalt, vooral wanneer personen aan sneller tempo onder de camera's lopen. Verdere tests en evaluatie met een gespecialiseerde dieptecamera behoren tot verder onderzoek.

Als laatste werd de performantie van de back-end getest. Hiervoor werden 1 065 035 fictieve

tellingsresultaten van 3376 trajecten van de Belgische treinoperator NMBS in een MySQL - tabel geplaatst. Vervolgens werd het vergelijken van verschillende trajecten en toepassen van filters getest. Wanneer het aantal te vergelijken trajecten beperkt blijft (ongeveer 15 trajecten per grafiek) blijft de rekentijd tevens binnen de perken (1 tot 15 seconden). Wanneer het aantal te vergelijken trajecten groter wordt stijgt de rekentijd drastisch (vanaf 40 trajecten loopt deze op tot 40 seconden) en worden de vergelijkingsgrafieken te druk om de datapunten te kunnen onderscheiden. De back-end is het beste geschikt om dagvergelijkingen van equivalent trajecten (trajecten met verschillende trajectnummers maar dezelfde set haltes) of week- of periodevergelijkingen uit te voeren van individuele trajecten (deze hebben een trajectnummer dat een traject op een bepaald moment van de dag represeneert).

Tot slot werden enkele ideeën voor uitbreiding en verder onderzoek opgeliist. Het implementeren van beeldcompressie, het toevoegen van extra configuratiegegevens aan camNode en het testen van het detectiealgoritme met gespecialiseerde hardware zijn de drie belangrijkste uitbreidingspunten. Desalniettemin werd een werkende en performante oplossing gevonden voor de uitdagingen en probleemstellingen die met de ontwikkeling van een APC - systeem voor treinen komen, waaronder schaalbaarheid, storingsbestendigheid en de dynamische samenstelling van treinstellen.

# Bibliografie

- [1] B. Barabino, M. Di Francesco en S. Mozzoni. "An Offline Framework for Handling Automatic Passenger Counting Raw Data". In: *IEEE Transactions on Intelligent Transportation Systems* 15.6 (dec 2014), p. 2443–2456. ISSN: 1524-9050. DOI: 10.1109/TITS.2014.2315573.
- [2] *Onderwerpen van IBCN*. URL: <https://www.ibcn.intec.ugent.be/sites/default/files/docs/2.%20PLATO%20masterproefonderwerpen%20AJ1617%20IBCN.pdf> (bezocht op 04-10-2016).
- [3] *BeluxTrains in het nederlands*. URL: <http://www.beluxtrains.net/indexnl.php?page=displaybeltrains> (bezocht op 28-04-2017).
- [4] S. Mukherjee e.a. "A novel framework for automatic passenger counting". In: *2011 18th IEEE International Conference on Image Processing*. 2011 18th IEEE International Conference on Image Processing. Sep 2011, p. 2969–2972. DOI: 10.1109/ICIP.2011.6116284.
- [5] *Zhengzhou Tiamaes Technology Co.,Ltd./TM8206 video passenger flow investigator*. URL: [http://en.tiamaes.com/Products/Hardware-series/The-vehicle-terminal-type/The-passenger-survey/49.html?gclid=CjwKEAjw6e\\_IIRDvorfv2Ku79jMSJAAuiv9YZR-jPOZtCUD5NqL20sUMsWt02756mZM\\_hCRytQ\\_bfhoCJhXw\\_wcB](http://en.tiamaes.com/Products/Hardware-series/The-vehicle-terminal-type/The-passenger-survey/49.html?gclid=CjwKEAjw6e_IIRDvorfv2Ku79jMSJAAuiv9YZR-jPOZtCUD5NqL20sUMsWt02756mZM_hCRytQ_bfhoCJhXw_wcB) (bezocht op 22-05-2017).
- [6] *3Doors Bus Passenger Counter*. URL: [http://www.vehicle-dvr.com/peoplecounterv3.html?gclid=Cj0KEQjwmIrJBRCRmJ\\_x7KDo-9oBEiQAUUPKMkrKV-Wk2BbyFtbv-wq0n1-NQfpBqp\\_bSJHo-\\_H9MLsaAm9o8P8HAQ](http://www.vehicle-dvr.com/peoplecounterv3.html?gclid=Cj0KEQjwmIrJBRCRmJ_x7KDo-9oBEiQAUUPKMkrKV-Wk2BbyFtbv-wq0n1-NQfpBqp_bSJHo-_H9MLsaAm9o8P8HAQ) (bezocht op 22-05-2017).
- [7] *Intel® RealSense™ Data Ranges / Intel® Software*. URL: <https://software.intel.com/en-us/articles/intel-realsense-data-ranges> (bezocht op 18-05-2017).
- [8] *Kinect for Windows Sensor Components and Specifications*. URL: <https://msdn.microsoft.com/en-us/library/jj131033.aspx> (bezocht op 01-06-2017).
- [9] *Kinect sensor, Ubuntu 14.04, NITE, OpenNI, install*. URL: [http://www.etti.tuiasi.ro/cin/Courses/SistEmbedded/Projects/Linux/BeagleBoard/Kinect\\_on\\_Ubuntu/Kinect\\_on\\_Ubuntu.html](http://www.etti.tuiasi.ro/cin/Courses/SistEmbedded/Projects/Linux/BeagleBoard/Kinect_on_Ubuntu/Kinect_on_Ubuntu.html) (bezocht op 01-06-2017).
- [10] *OpenNI 2 Downloads and Documentation / The Structure Sensor*. URL: <https://structure.io/openni> (bezocht op 01-06-2017).
- [11] *Intel® RealSense™ Developer Kit (R200), VF0830*. URL: [https://click.intel.com/intelr-realsense\(tm\)-developer-kit-r200-2248.html](https://click.intel.com/intelr-realsensetm-developer-kit-r200-2248.html) (bezocht op 30-05-2017).
- [12] *opencv/opencv*. GitHub. URL: <https://github.com/opencv/opencv> (bezocht op 09-05-2017).
- [13] *Servus: Main Page*. URL: <https://hbpvis.github.io/Servus-1.4/index.html> (bezocht op 20-03-2017).
- [14] *GStreamer: open source multimedia framework*. URL: <https://gstreamer.freedesktop.org/> (bezocht op 31-05-2017).
- [15] *Multiple streaming in c++ using opencv; OpenCV video streaming over TCP/IP*. Gist. URL: <https://gist.github.com/Tryptich/2a15909e384b582c51b5> (bezocht op 30-11-2016).
- [16] *Distributed Messaging - zeromq*. URL: <http://zeromq.org/> (bezocht op 30-03-2017).

- [17] *The World's Best Photos of 307 and arriva - Flickr Hive Mind*. URL: <https://hiveminer.com/Tags/307,arriva/Interesting> (bezocht op 29-05-2017).
- [18] John WH Smith. *Capturing a webcam stream using v4l2*. JWH Smith. 3 dec 2014. URL: <https://jwhsmith.net/2014/12/capturing-a-webcam-stream-using-v4l2/> (bezocht op 30-11-2016).
- [19] *3D-modellen*. URL: <http://www.seos-project.eu/modules/3d-models/3d-models-c02-p04.nl.html> (bezocht op 08-05-2017).
- [20] *LEADING EDGE VIEWS: 3-D Imaging Advances Capabilities of Machine Vision: Part I*. URL: <http://www.vision-systems.com/articles/print/volume-17/issue-4/departments/leading-edge-views/3-d-imaging-advances-capabilities-of-machine-vision-part-i.html> (bezocht op 09-05-2017).
- [21] *What Is Camera Calibration? - MATLAB & Simulink - MathWorks Benelux*. URL: <https://nl.mathworks.com/help/vision/ug/camera-calibration.html?requestedDomain=www.mathworks.com> (bezocht op 09-05-2017).
- [22] *Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix ←*. URL: <http://ksimek.github.io/2013/08/13/intrinsic/> (bezocht op 09-05-2017).
- [23] *camera\_calibration/Tutorials/MonocularCalibration - ROS Wiki*. URL: [http://wiki.ros.org/camera\\_calibration/Tutorials/MonocularCalibration](http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration) (bezocht op 20-11-2016).
- [24] *opencv: Open Source Computer Vision Library*. original-date: 2012-07-19T09:40:17Z. 10 nov 2016. URL: <https://github.com/opencv/opencv>.
- [25] *Stereo Vision Tutorial - Part I · Chris McCormick*. URL: <http://mccormickml.com/2014/01/10/stereo-vision-tutorial-part-i/> (bezocht op 09-05-2017).
- [26] *socket(2) - Linux manual page*. URL: <http://man7.org/linux/man-pages/man2/socket.2.html> (bezocht op 01-06-2017).
- [27] *NMBS - Dienstregeling, Biljetten, Abonnementen en Tarieven*. URL: <http://www.belgianrail.be/nl> (bezocht op 01-06-2017).
- [28] *curlpp by jpbarrette*. URL: <http://www.curlpp.org/> (bezocht op 01-06-2017).
- [29] *PHPixie - Modern, Fast and Secure PHP Framework*. URL: <https://phpixie.com/> (bezocht op 01-05-2017).
- [30] *Symfony, High Performance PHP Framework for Web Development*. URL: <https://symfony.com/> (bezocht op 01-05-2017).
- [31] *OAuth Community Site*. URL: <https://oauth.net/> (bezocht op 01-05-2017).

# **Hoofdstuk 10**

## **Bijlagen**

### **10.1 GitHub repository**

De broncode voor camNode, camServer, de back-end, de emulator voor de boordcomputer en de testnode, alsook een dump van de SQL - tabellen gebruikt in de back-end en een testfilm met dieptebeelden zijn op een GitHub repository geplaatst. Deze repository is beschikbaar via onderstaande URL.

*<https://github.com/lorre851/thesis>*

