TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

EETU KAUHANEN

AN APPROACH TO AUTONOMOUS COLLISION AVOIDANCE IN A MONORAIL TRANSFER SYSTEM

Master's thesis

Examiner: professor Jose M. Lastra
Examiners and topic approved by the Faculty Council of the Engineering Sciences on January 2017

ABSTRACT

EETU KAUHANEN: An approach to autonomous collision avoidance in a Monorail transfer system
Tampere University of Technology
Master of Science Thesis, 50 pages, 3 Appendix pages
July 2017
Master's Degree Programme in Automation Technology
Major: Factory Automation and Industrial Informatics
Examiners: Professor Jose M. Lastra

Keywords: collision avoidance, monorail, intralogistics, factory automation, sensors, intelligent transportation systems, Node.js, Modbus, GUI

Collision Avoidance Systems are utilized both in industry and traffic in attempt to prevent material losses and injuries. These systems are implemented using sensors, communication systems or a combination of these. The most used sensors in collision avoidance applications are optical, electromagnetic and ultrasonic sensors. The communication-based systems use both wireless and wired communications utilizing various protocols.

This document describes the process of creating a prototype of a sensor-based Collision Avoidance System for Cimcorp Monorail Transfer system. Monorail Transfer is an automatic transportation system used in tire manufacturing plants for moving the tires between different process stations. It consists of a rail, which is either fastened into the roof of the plant or into a leg-like support structure, carriers which move along the rail and a cell controller. The aim is to prevent the collisions between the carriers and between carriers and other objects. The aim of the work is to have a functional prototype of an autonomous, sensor-based Collision Avoidance System which does not depend on Wi-Fi.

The work begins with introducing the concept of Monorail Transfer in detail. Next the technologies behind the existing collision avoidance systems in industry and traffic are reviewed. The sensor types used in the implementations are identified and reviewed. Their suitability for the Monorail is considered. It is found that electromagnetic and optical sensors would be most suitable for the system. Electromagnetic sensors are discarded due to their high price and power consumption. Communication-based systems are reviewed. The Monorail Transfer's current Collision Avoidance System is studied.

After the theoretical part the new system is designed after defining its requirements. The sensors are chosen and reviewed. A Raspberry Pi 2 model B is chosen for pre-processing the sensor data prior to introducing it to the PLC's in the Monorail Transfer. The Collision Avoidance software is programmed in Node.js, the communications between the PLC and Raspberry Pi are programmed in Node.js and the Graphical User Interface is implemented using HTML5, CSS and JS.

The system is tested thoroughly and modified according to the findings. The prototype is found to be functional. Finally the document ends with conclusions about the implemented prototype.

TIIVISTELMÄ

EETU KAUHANEN: Lähestymistapa autonomiseen törmäyksenestoon Monorail-kuljetinjärjestelmässä
Tampereen Teknillinen Yliopisto
Diplomityö, 50 sivua, 3 liitesivua
Heinäkuu 2017
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Factory Automation and Industrial Informatics
Tarkastaja: Professori Jose M. Lastra

Avainsanat: törmäyksenesto, Monorail, sisälogistiikka, tehdasautomaatio, anturit, älykkäät kuljetusjärjestelmät, Node.js, Modbus, GUI

Törmäyksenestojärjestelmiä käytetään sekä teollisuudessa että liikenteessä vähentämään törmäyksistä aiheutuvia henkilö- ja aineellisia vahinkoja. Järjestelmiä toteutetaan antureita, tiedonsiirtojärjestelmiä tai näiden yhdistelmiä hyödyntäen. Törmäyksenestojärjestelmissä useimmin käytetyt anturityypit ovat optiset, sähkömagneettiset ja ultraääniantu-rit. Tiedonsiirtoon pohjautuvia järjestelmiä toteutetaan sekä langatonta että langallista tiedonsiirtoa käyttäen.

Tämä dokumentti kuvaa anturipohjaisen törmäyksenestojärjestelmän prototyypin luomisprosessin Cimcorpin Monorail Transfer- järjestelmää varten. Monorail Transfer on automaattinen kuljetinjärjestelmä, joka kuljettaa renkaita rengastehtaassa prosessiasemalta toiselle. Se koostuu kiskosta, vaunuista jotka liikkuvat kiskoa pitkin ja kiskon tukirakenteesta. Tavoitteena on estää vaunujen törmääminen sekä toisiinsa että ulkopuolisiin esteisiin. Tämän työn päämääränä on luoda toimiva prototyyppi autonomisesta, anturipohjaisesta ja Wi-Fi-riippumattomasta törmäyksenestojärjestelmästä.

Työ alkaa tarkemmalla Monorail Transferin esittelyllä. Seuraavaksi esitellään teollisuuden ja liikenteen törmäyksenestojärjestelmien takana olevaa teknologiaa. Toteutettujen järjestelmien anturityypit todetaan ja niiden soveltuvuutta Monorailiin tutkitaan. Huomataan, että sähkömagneettiset ja optiset anturit soveltuvat tähän toteutukseen parhaiten. Sähkömagneettiset anturit jätetään pois laskuista niiden korkean hinnan ja energiankulutuksen vuoksi. Tiedonsiirtojärjestelmiin pohjautuvia järjestelmiä esitellään. Monorail Transferin tämänhetkistä törmäyksenestojärjestelmää tutkitaan.

Teoreettisen osion jälkeen suunnitellaan uusi järjestelmä määriteltyihin vaatimuksiin pohjautuen. Anturit valitaan ja esitellään. Raspberry Pi 2 Model B valitaan prosessoimaan dataa ennen kuin se viedään PLC:lle. Törmäyksenestoalgoritmi ja tiedonsiirto PLC:n ja Raspberry Pi:n välillä ohjelmoidaan Node.js:llä. Graafinen käyttöliittymä luodaan käyttäen HMTL5:ttä, JS:a ja CSS:ä.

Järjestelmää testataan läpikotaisin ja muokataan löydösten perusteella. Prototyyppi todetaan toimivaksi. Lopuksi yhteenvedossa koostetaan toteutuksen tulokset.

PREFACE

This Master of Science thesis was made for Cimcorp Oy, Ulvila as a product development project during spring 2017. I wish to thank Cimcorp for providing me an interesting topic for my thesis as well as all the needed materials.

I'd like to thank Mr. Jyrki Anttonen for helping me to get started with the work. Thanks for Mr. Jani Tuomola for providing his valuable expertise about the Monorail Transfer when needed.

This work was somewhat challenging but rewarding, as I gained massive amounts of valuable experience in the field of my major studies.

Thanks for prof. Lastra at Tampere University of Technology for supervising this thesis.

Special thanks for Mr. Aku Karhunen for a drunk phone call in 2010, persuading me to apply to TUT.

Thanks for my wife for the support she has given me during this intense period. $< |\sqrt{9}|$

Thanks for my daughter and son for keeping my thoughts off the thesis while at home.

Ulvila, 24.07.2017

Eetu Kauhanen

TABLE OF CONTENTS

TERMS AND ABBREVIATIONS

CAS	Collision Avoidance System
CPU	Central Processing Unit
FoV	Field of Vision
F_{tot}	Total force, sum of all forces
GPIO	General-Purpose Input/Output
GUI	Graphical User Interface
ITS	Intelligent Transportation Systems
LADAR	Laser Sensing and Ranging
LAN	Local Area Network
LIDAR	Light Detection and Ranging
npm	node package manager
PLC	Programmable Logic Controller
RADAR	Radio Detection and Ranging
RasPi	Raspberry Pi
RS-485	Recommended Standard 485
RTU	Remote Terminal Unit
TOF	Time-of-flight
V2I	Vehicle-to-infrastructure
V2V	Vehicle-to-vehicle
VFF	Virtual Force Field

1. INTRODUCTION

This thesis is made for Cimcorp Oy as a product development project. The aim of this work is to develop a prototype of a more autonomous collision avoidance system for Cimcorp's Monorail transfer system.

Cimcorp Monorail transfer system is an automatic transport system used in tire industry for transferring both raw and finished tires between different process stations. This chapter introduces the Monorail to an extent that is necessary for comprehending the scope of this work.

1.1 Structure of the Monorail Transfer System

The basic scope of delivery for the Monorail transfer consists of three main components: the carriers, the rail, and the rail support structure. Out of the three parts the rail and the carrier are relevant considering the scope of this work. Their structure and functions are considered in more detail in following subchapters.

1.1.1 Rail

The rail acts naturally as a mechanical support structure for the carrier. The shape of the rail determines the direction of movement of the Monorail carrier. Besides of this the rail includes lines for electricity transmission, signal pathways for safety functions and a pulse encoder rail for positioning. The rail is situated in the roof level of the factory, either fixed into any support structure which fulfills certain criteria or into a separate, floor-fixed leg support structure.

The rail lies in one horizontal plane, but it may have curves among the way. There are three basic configurations provided:

- Bidirectional line. This consists of a straight line between the process stations. Usually this configuration has two carriers per line, moving in both possible directions. The carriers can have designated sections. There may be as many lines in a factory as needed.
- Bidirectional line and curve. This consists of a straight line with a curve in another end. This configuration has typically two carriers per line, both carriers having their own section.
- One-directional loop. This consists of a rectangular loop with curves in each corner. This configuration may have as many carriers as needed. The carriers in the

loop may travel only in one direction as the name proposes. This is the most important configuration concerning this work.

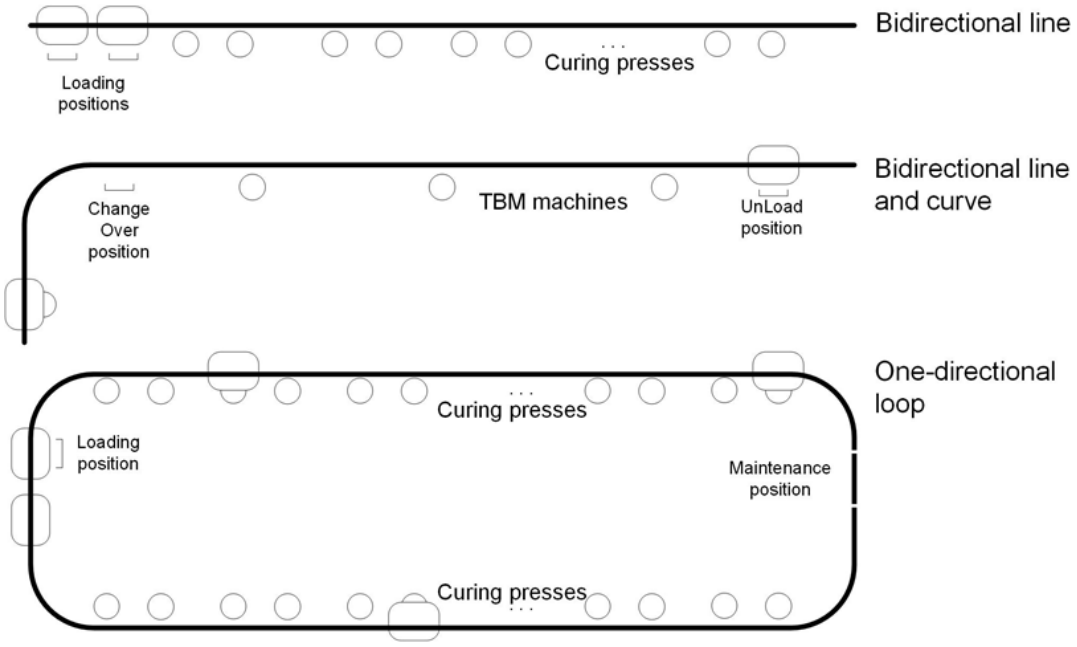


Figure 1: *Possible rail configurations, view from above.*

Figure 1 illustrates the aforementioned configurations from above. As seen on the figure, there are usually multiple positions where the carrier may pick or drop the tire.

The curves in the rail provide one of the main challenges of this work, which is detecting objects around the corner, especially when there might be any solid support structures on the way. This applies especially to the One-directional loop rail configuration. The curves may also be tight, for the allowed minimum bend radius is 1 m.

1.1.2 Carrier

The carrier is used for picking up tires from the loading position, lifting them up and carrying them to any process station, where they are lowered back to the floor level and dropped out from the carrier. The movement of the carrier is achieved with electrical servo motors. The carrier is powered by the power line enclosed in the rail. The picture below reveals the main parts of the carrier.

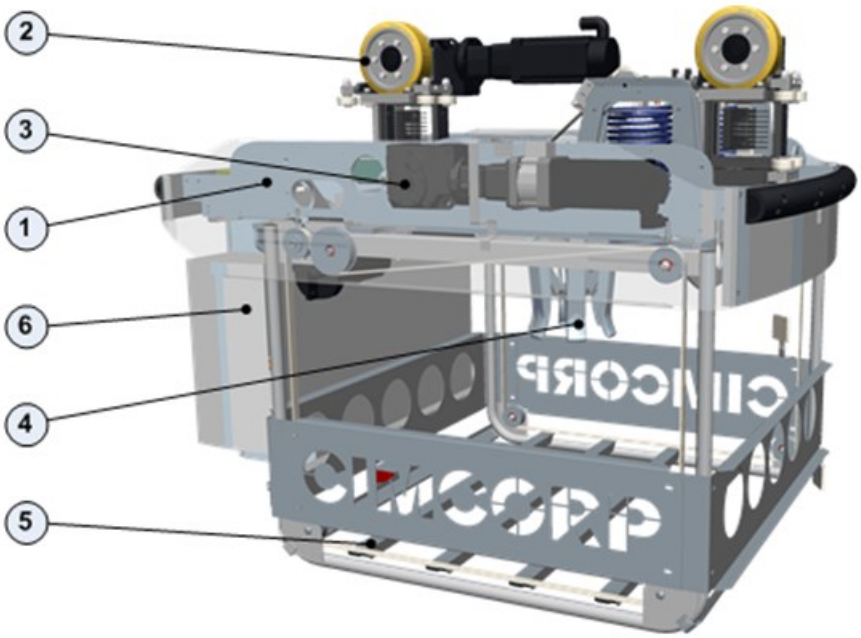


Figure 2: *Monorail carrier.*

The parts, as numerated in Figure 2, are the following:

1) The carrier frame
2) Drive module for horizontal movement
3) Drive module for vertical movement
4) The tire gripper
5) The safety hatch
6) The onboard control cabinet.

The frame offers a platform for installing the other devices and protects the tire. Drive modules are for moving the carrier among the rail and lowering and hoisting the gripper. The tire gripper is for gripping the tires and the safety hatch is for preventing the tires from falling off from the carrier. The onboard control cabinet includes the programmable logic controllers, servo drives, I/O-modules, fuses and general electricity equipment.

The programmable logic controllers or *PLC*'s in the cabinet are manufactured by Rockwell. They are programmable through a USB-interface and support various communication protocols. The preferable protocol here is Ethernet/IP and the programming language used for the programming is any of the languages defined in the IEC 61131-3.

The carrier travels on the rail with a velocity up to 5 m/s. The weight of the carrier is 680 kg while empty. Maximum payload for the carrier is 120 kg.

According to tests performed at Cimcorp, the distance between the starting point of braking and the position where the carrier stops is approximately 7 m. Thus, it is a matter of

uttermost importance to detect any obstacles on the way well before a collision would happen for avoiding material and personal losses as well as delays in production.

1.2 Control system

The Monorail system is controlled in multiple levels. Each carrier has its own control cabinet containing servo drives and PLCs, while the robot cell consisting of one loop or rail is controlled by a cell controller. The cell controller itself receives commands from higher-level host systems. The cell can also be controlled to some extent via the Cell user interface.

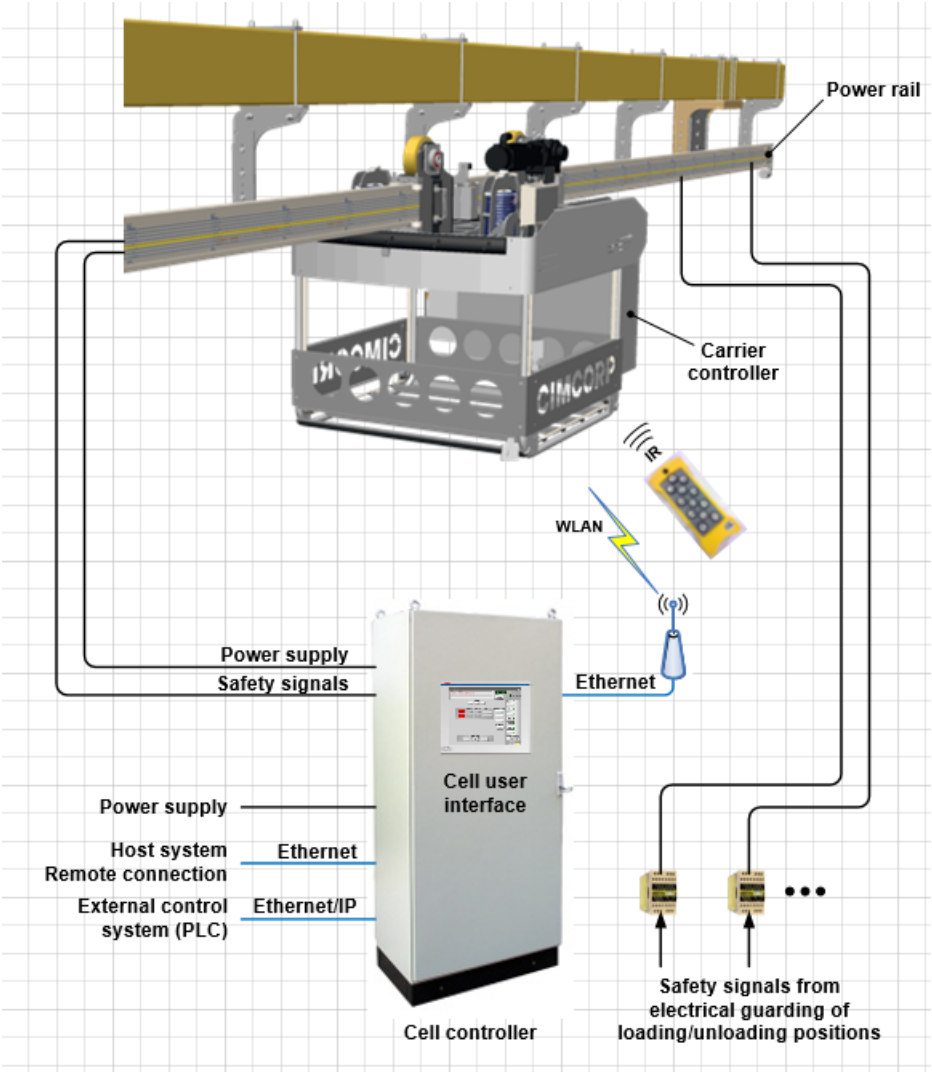


Figure 3: *High-level overview of the Monorail control system.*

Figure 3 illustrates the control system at high level. Connections to host system are implemented via Ethernet, connections to external control systems via Ethernet/IP and the communication between separate carriers and the cell controller via Wi-Fi. Some safety

signals from factory floor level are brought within the rail to ensure their passage to each carrier. Currently the carriers communicate with each other passing the messages (for example their positions for avoiding collisions) via the cell controller, which acts as a message broker. The Wi-Fi based connection is anyhow vulnerable and prone to connection losses and delays [15, p. 7], [16, p. 333], [41, ch. 2, p. 2] leading to the need of creating a more autonomous collision avoidance system.

1.3 The challenges in the current system

The current collision avoidance system in the Monorail transfer is a fusion of sensor-based perception and inter-device communications. The carriers are aware of each other's positions, but in cases where the connection is either lagging or lost, the collision avoidance system relies wholly upon the sensory implementation. The sensory implementation has poor performance in the proximity of the curves in the rail, since the sensors cannot currently sense objects behind the curve. The current system is also unable to detect objects below the carrier's path, for example personnel lifts or other hoisting devices. Thus, the system needs some re-implementation.

1.3.1 Sensor-based implementation

The sensory implementation consists of only one single-beam laser sensor per carrier measuring the distance to the next carrier on the rail. The sensor has a narrow *Field-of-View* or *FoV*, so it cannot detect anything else than the carrier on the front. The sensor is in a fixed position, so it cannot detect obstacles when the carrier is either in a curve, approaching a curve or leaving the curve.

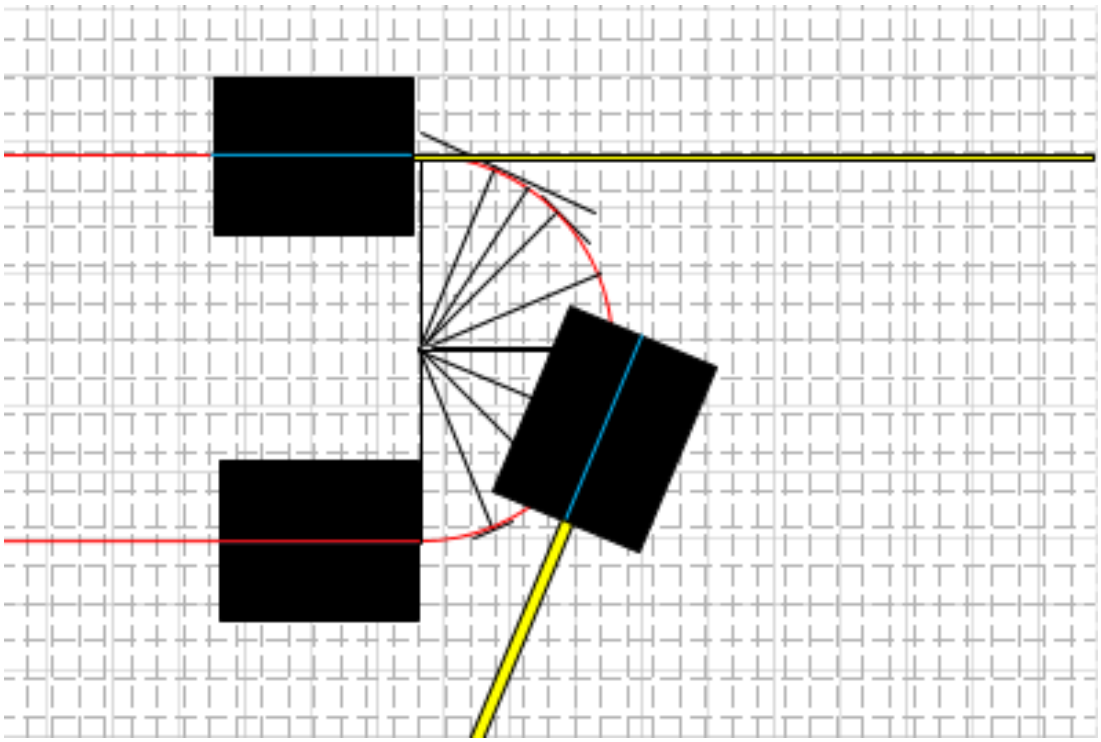


Figure 4: *An example situation.*

Figure 4 illustrates the aforementioned issue. The red curve is the rail, the black blocks are the carriers and the yellow bars are the beams. None of the sensors are detecting the next carrier. In the case of a connection failure this could lead to a collision, which in turn could lead to personal or material damage and delays in production.

1.3.2 Communication-based implementation

The Monorail carriers are aware of their absolute position thanks to a linear encoder. They communicate with each other through the Cell controller, forming a client-message broker-pattern, in which all the carriers are aware of the other carriers' positions. The communication is implemented with Wi-Fi. Wi-Fi is known to have some issues [15, p. 7], [16, p. 333], [41, ch.2, p. 2]:

- Wi-Fi uses unlicensed radio spectrum, leading to the situation where other protocols may interfere with Wi-Fi [16, p. 333].
- Power consumption of Wi-Fi is relatively high [16, p. 333].
- The maximum length for the Monorail transfer loop is 300 m, leading to long distances between the Cell controller and the carriers. Longer distance leads to a weaker connection and increased latency.
- Issues in interoperability between different brands may lead into increased latency and connection losses [16, p. 333].

- In a typical installation site there will be obstacles between the Wi-Fi access point and the clients (carriers).
- In cases of connection losses, the system is susceptible for collisions, especially in the proximity of the curves.

One approach for improving the CAS would be improving the wireless connection, but it was the company's wish to introduce a more autonomous, less Wi-Fi-dependent system, since there's better use for the Wi-Fi-connection.

Next the document proceeds with the results of the research work concerning currently existing collision avoidance systems and the technologies behind them. The main focus is on existing implementations in traffic and industry. Sensor technology is reviewed as finding suitable sensors was one of the goals in the work.

After the research part, the document reveals the proposed prototype of the system. The used devices and the implemented software are studied. The created Graphical User Interface is reviewed. The communications between the prototyping platform and the existing data infrastructure are presented.

Next the document proceeds with validating the created prototype. Testing procedures as well as results are presented for the core software and for the Graphical User Interface. The modifications done after testing are listed. The findings are concluded.

Finally, the document ends with conclusions and thoughts about future development. The strengths and weaknesses of the new implementation are considered. The changes needed for turning the prototype into an actual product are reviewed.

2. LITERATURE REVIEW

Developing *collision avoidance systems* or *CAS* has been an interest of the automatized manufacturing industry for the last three decades. Efficient collision avoidance could decrease the amount of machine-related accidents and thus decrease the risk for injuries and material losses. Automotive industry has adopted many of these technologies, developing them even further [6, p. 1175]. This chapter describes the collision avoidance systems and technologies used both in industry and in traffic and provides an overview to the current collision avoidance implementation in the Monorail transfer.

2.1 Technologies

Even though there are countless possible technological solutions for creating a collision avoidance system, most of the studied systems tend to follow one or more of the following principles:

1) Sensor-based perception
2) Inter-device communications
3) Machine Vision [1].

The sensor-based ones use sensors to obtain data from their environment, while the inter-device communication-based systems just keep each other informed about their positions [1]. Currently the Monorail transfer's collision avoidance system is a fusion of both, though the sensory part is poorly implemented.

2.1.1 Sensors

For the whole history of active collision avoidance systems the sensors used for collision avoidance and obstacle detection have been limited to a set of three [8],[1],[2],[13, p. 322]. These are ultrasound, electromagnetic and infrared sensors. This chapter presents some of their key features and differences.

Ultrasound sensors are popular due to their low price and power consumption [21, p. 41]. The operational principle of an ultrasound sensor is the following:

1) The sensor emits high-frequency (over 20 kHz) sound
2) The sound is reflected from a surface
3) The sensor detects the echo and calculates the return time [4, p. 273], [5, p. 341].

Using the return time, the distance can be calculated as the speed of sound is known. This is widely known as *Time-Of-Flight* or TOF ranging [6, p. 501].

Besides of the distance measurements, ultrasound sensors can be used for measuring movement and velocity utilizing the Doppler phenomena [19, p. 352].

Ultrasound sensors have a broad field of detection, they are light-weight and cheap. They can detect transparent items as well as shiny objects. Their accuracy is around ±1 mm [5, p. 341]. Their sensing range tends to be limited to a maximum of 6 meters [5, p. 341]. The velocity of sound through a medium is dependent on ambient temperature [5, p. 631], so ultrasound sensors are usually unsuitable for environments with high ambient temperatures [4, p. 130], [5, p. 342]. The sensing rate is lower compared to infrared sensors due to the difference between the speeds of sound and light [6, p. 493].

Infrared sensors as well as other optical sensors have multiple different ways of usage. One of the most utilized ways is using a sender-receiver pair: the sender emits a light pulse and the detector absorbs it. If there's an obstacle between the pair, the passage of the light will be interrupted and the light will not be absorbed at the receiver. This counts for a detection. The method is called "through scanning" [5, p. 361], [19, p. 341].

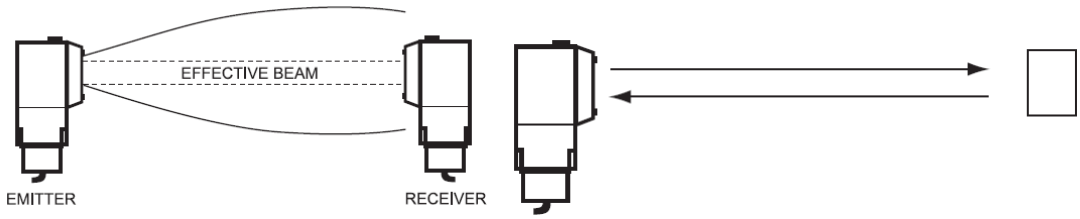


Figure 5: *Different usages of optical sensors [5, p. 361].*

Another popular way is enclosing both into same closure and measuring reflected light. Also here the TOF ranging can be used [10, p. 233], [39, p. 86]. This method is called "diffuse scanning" [19, p. 341]. Figure 5 illustrates the difference between the two scanning methods. Out of these methods the diffuse scanning would look more suitable for obstacle detection necessary for collision avoidance, since the objects that the system needs to detect cannot always lie between the emitter-receiver pair.

Infrared sensors have a broad detection range. They detect objects from as near as 0.1 µm to as far as 100 m and even further [5, p. 358]. They may use a single point of light for measurements [9, p. 2320] or have a spread pattern of light, which can be read in segments. They have a faster sensing rate compared to the ultrasound sensors due to the difference between the speeds of sound and light. Infrared sensor are a bit more expensive than ultrasound sensors. They are immune to background radiation and electromagnetic fields [4, p. 362].

However most of the traditional infrared sensors are sensitive to ambient light, weather conditions and heat [4, p. 156]. The quality of the detection is somewhat dependent on

the optical qualities of the object [19, p. 344]. The sensitivity to heat can on the other hand be advantaged in object recognition: for example, humans emit infrared waves inside a narrow spectrum, providing the possibility to identify humans using their natural temperature fingerprint [4, p. 156], [11, p. 102].

Other widely used optical sensors use lasers and even LEDs within or outside of the range of infrared [4, p. 299]. These are also used both in industry and traffic. Lasers are quite broadly implemented in scanner-type sensors, which can detect objects in a 2D-plane or even in 3D [9, p. 2324]. This kind of equipment can be used for generating maps of environment as well as localizing objects [9, p. 2324]. These devices are usually called LADARs or LIDARs, standing for *"Laser Detection And Ranging"* and *"Light Detection And Ranging"*, respectively.

Due to the wide-field perception obtained with scanner type sensors, they are broadly used in moving equipment. Some of these products divide their Field-of-View or *FoV* into separate segments, which allows a more precise localization of the detected objects [4, p. 297].

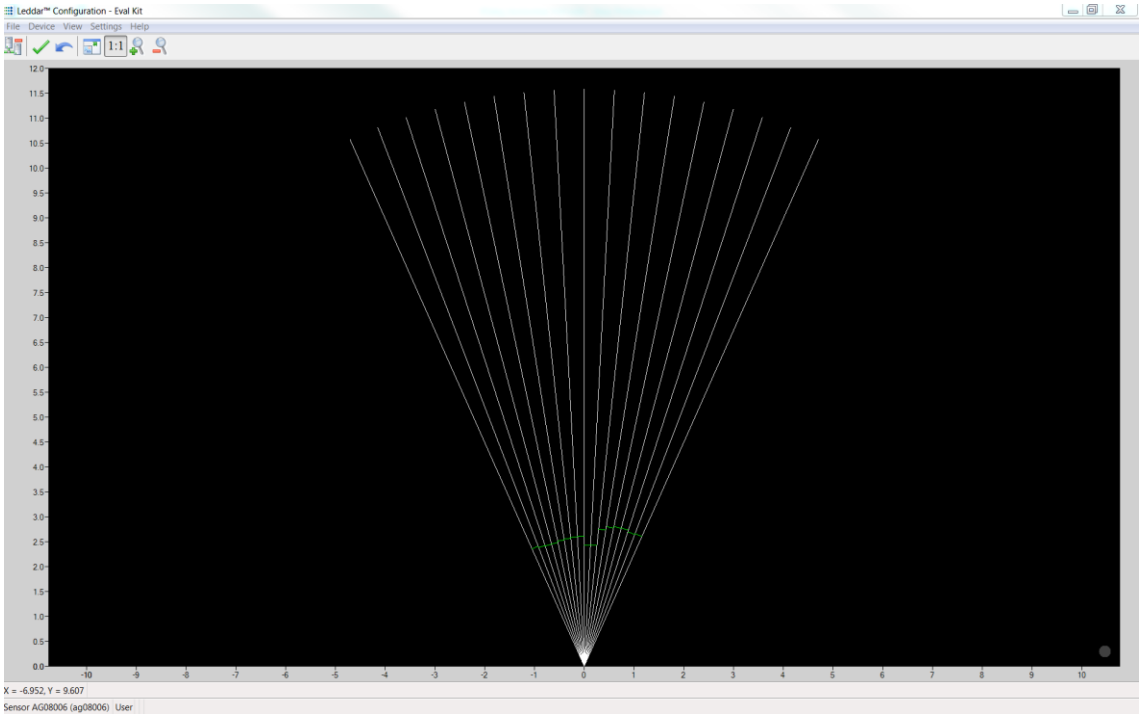


Figure 6: *A multi-segment LED scanner detecting objects.*

Figure 6 provides an example of a multi-segment LED sensor detecting some objects in its near proximity. The figure was produced experimentally using a LEDDARTech M16-sensor. The sensor was lying flat on a desk while capturing the image. The notch seen in the middle segments represents a structure extruding from the ceiling.

Electromagnetic sensors refer in this context to RADAR, or *"Radio Detection And Ranging"* [4, p. 277]. Their operational principle is somewhat similar to the previous ones: the

sensor emits a pulse of electromagnetic radiation (in the microwave spectrum), the pulse is reflected from a surface and partially captured by the sensor's detector. TOF calculations are performed to determine the location of the object [11, p. 102]. Radar can also distinguish the object's size and shape to some extent [12, p. 10-11].

Radars are virtually immune to any environmental conditions [11, p. 102]. They have a range from micrometers to kilometers. Radars can detect an object independent of the object's material. Radars have the second-highest sample rate in this comparison [12, p. 18], [4, p. 277]. Their resolution is dependent on the operation frequency [14, p. 29].

Price of radars is higher than the other sensors'. They have the highest power consumption in this trio.

The following table provides a quick summary of the key features of the three sensor types.

Table 1: *Quick comparison between the sensor types.*

Sensor type	Detection range	Frequency	Price	Power consumption	Resolution	Object size	Sensitivity to environmental conditions
Ultrasound	1 mm – 6 m	20 kHz – 200 MHz	€	Low	1 mm	cm	Temperature
Optical	1 µm – 100 m	300 GHz – 430 THz	€€	Middle	0.1 mm	mm	Temperature, humidity
Electromagnetic	1 mm – millions of km	30 Hz – 300 GHz	€€€	High	1 mm	mm	None

2.1.2 Inter-device communications

Another approach to collision avoidance is based on communication between the objects prone for collision. This kind of systems depend either on wireless communications between devices, wireless communications between devices and a message broker or wired communications between the participants [3, p. 1]. Both methods are used both in industry [6, p. 925] and traffic [3, p. 1], though the wireless methods quite obviously dominate the vehicular use.

2.1.3 Machine Vision

In this context, the term *"Machine Vision"* refers to the usage of monocular and stereo cameras for providing images of the environment [28, p. 1]. The data obtained by the cameras is usually heavily processed to extract certain distinct features. Machine vision

can be utilized using color [29, p. 133], shape, edges [2, p. 458] or various other predefined features as reference [27, p. 25, 53, 73, 111]. Some of the approaches utilize the lights of the other vehicles for recognizing them and determining their behavior [30, p. 451].

Machine vision equipment can be harnessed also to simple tasks such as detection and measurement [32, p. 529]. Anyhow the full potential lies in the capability to recognize objects, giving opportunity to adjust behavior based on the nature of the detection.

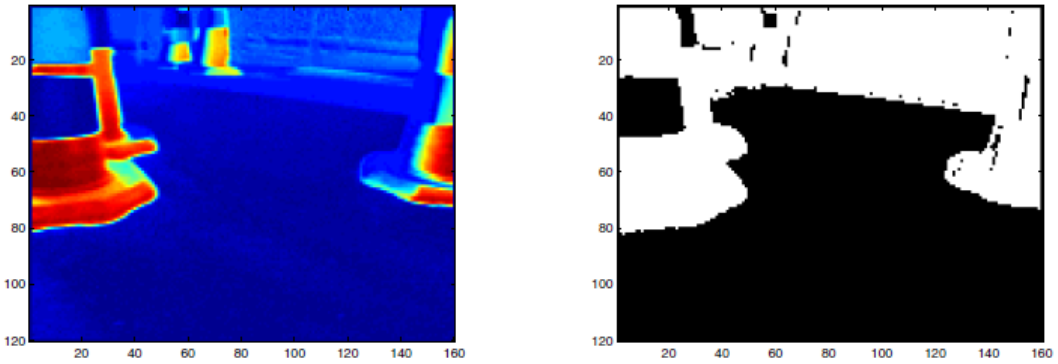


Figure 7: *An example of distance measurement and obstacle detection with a stereo camera [32, p. 529].*

Figure 7 provides a visual example of camera-based measurements and detections. The distance measurements are based on light intensity. Detections are based on the same data excluding the depth information.

Object recognition follows typically the following pattern [27, p. 460]:

- Object's features are detected
- A hypothesis about the nature of the object is formed upon the features
- The features are compared against a model in the database according to the hypothesis
- The hypothesis is either verified or discarded according to the result of the comparison.

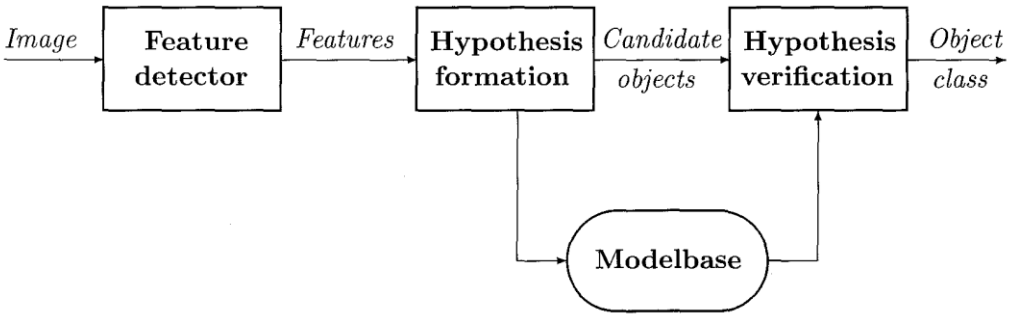


Figure 8: *Object recognition pattern [27, p. 460].*

Figure 8 illustrates the process of object recognition. The detected features may be any of the ones mentioned earlier or a combination of them.

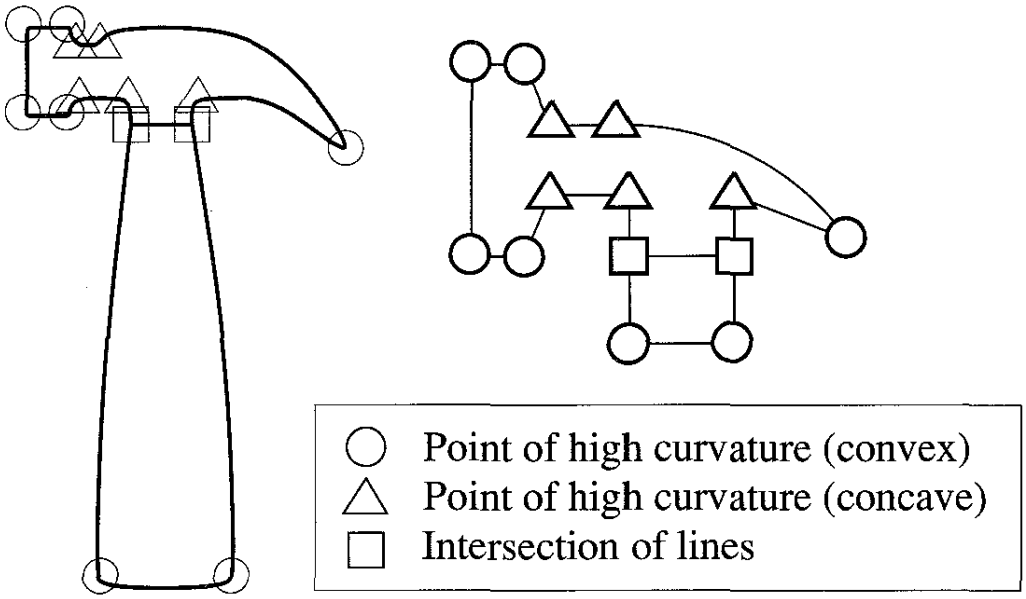


Figure 9: *An example of the features [27, p. 468].*

Figure 9 provides an example of what the detected features might be. In this example the features are absolute or local features [27, p. 468]. Other type of features are relative features, as in the following figure.

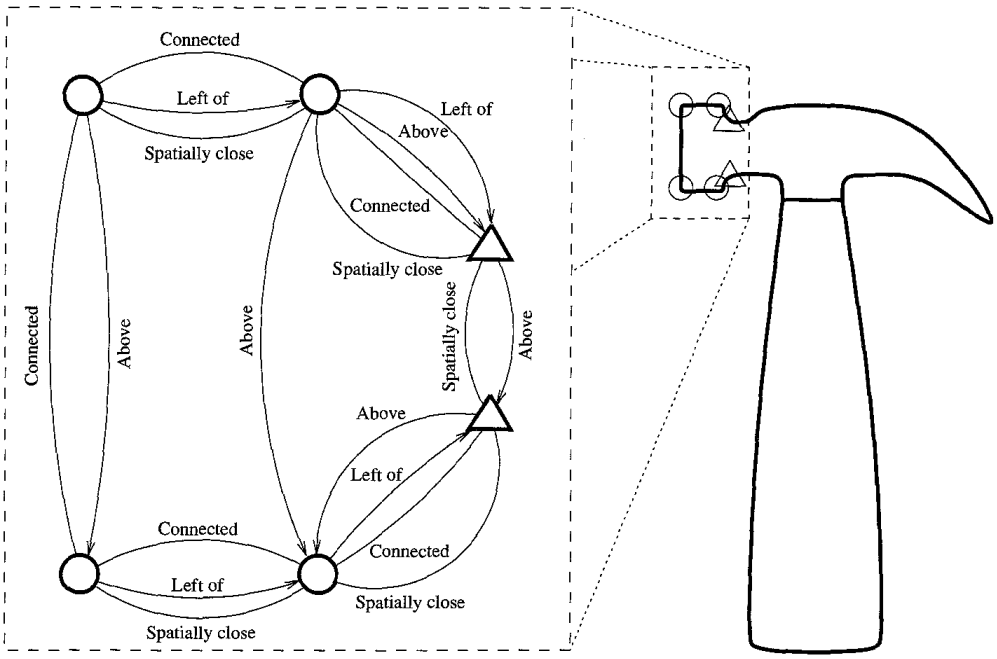


Figure 10: *Example of relative features [27, p. 473].*

Figure 10 illustrates the concept of relative features. Instead of just reading geometrical properties of an object, some of the most distinctive points are linked together using their relations.

By combining both relative and absolute features, more complex shapes can be constructed. Thus, complex models of the objects can be built before comparing then to the model database [27, p. 468]. Later chapters present the usage in traffic applications and robotics in more detail.

2.2 Collision avoidance in automotive industry and rail-bound traffic

Since the automotive industry has been developing collision avoidance technologies for decades, having a review at the automotive implementations is justified when developing a new system. This subchapter provides a quick look on the collision avoidance systems used in traffic.

2.2.1 Sensor-based implementations

Most of the current sensor-based automotive collision avoidance systems are based on fusion of multiple sensors of different types [3, p. 1]. They may be placed all around the vehicle to create a 360° view or in a more centralized manner to provide redundancy by partial overlapping. The sensor types used in vehicles are the same as in collision avoidance systems overall: optical, electromagnetic and ultrasound [3, p. 1], [17, p. 2004], [21, p. 196]. Since automotive collision avoidance is a matter of safety, the system cannot rely upon a single sensor, so multiple sensors are applied also for this reason [21, p. 663].

Another reason for using multiple sensors simultaneously is to detect and track multiple objects in the same time, as in [17]. The study, among many others, describes the usage of infrared and radar sensors together. The infrared sensor detects objects on a wide area, while the radar concentrates more on detecting objects further away with a narrow focus. The sensors' areas overlap in the front-facing section, providing redundant detections from the direction of most destructive collisions [17, p. 2004].

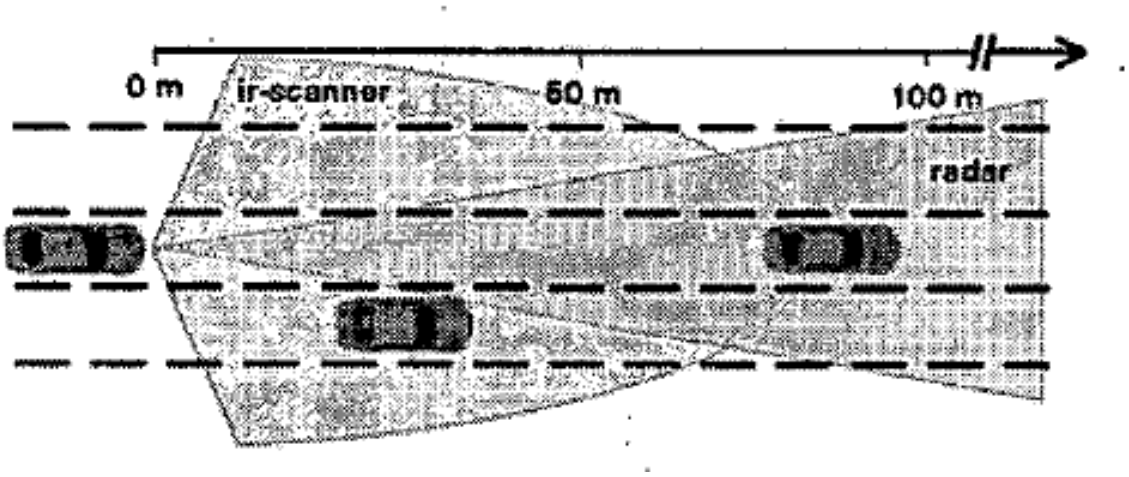


Figure 11: *An example of sensor fusion [17, p. 2004].*

Figure 11 illustrates the example. Radar detects the car which is driving further away, while the infrared sensor detects the car driving right to the sensor-equipped vehicle. The radar alone would not detect the latter, and the IR itself would neglect the first.

The usage of different types of sensors allows for detection of objects of different sizes and types. For example electromagnetic sensors can detect smaller objects than ultrasound sensors [18, p. 184] and infrared sensors can measure distance also in near proximity while the radars can't. Radars and ultrasound sensors can detect transparent objects which the optical sensors would ignore [19, p. 344], [9, p. 2057].

Ultrasound sensors are utilized in vehicles mainly for parking assistant systems and other near-proximity collision avoidance systems [3, p. 2], [18, p. 183]. Their sensing rate is

slower than the other two types and their range is significantly shorter than the range of the others. Thus they cannot be relied upon in safety-critical systems [3, p. 2].

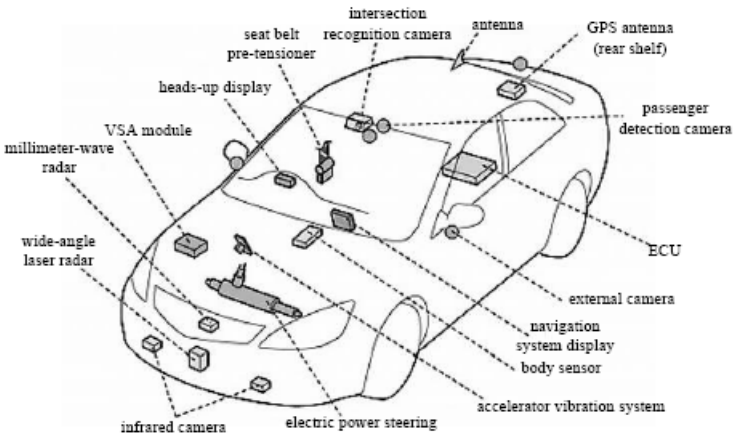


Figure 12: *Honda ASV-3 equipped with multiple intelligent systems [18, p. 185].*

Figure 12 shows typical setup of sensors for collision avoidance: millimeter-wave radar for long distance obstacle detection [21, p. 66-67] and a wide-range laser radar for detecting obstacles on a wider area. In this system, the infrared cameras in the front bumper are for providing the driver a live image of the environment while driving in darkness.

Since the aim of this work is to create a CAS for a rail-bound device, having a look at CAS implementations for railroad equipment appears intuitive.

Some development work with laser scanners has been evaluated. For example [33] presents a system, where a tram is equipped with a bunch of front-facing laser scanners with wide FoV's. Here the width of the FoV should cover all the curves among the way, since railroad curves cannot be extremely tight due to the length of the vehicles. The system utilizes also a "road map" for the route to be driven in order to neglect all the detections that would occur from objects which belong to the sides of the route [33, p. 727]. The same work as well as others also highlights some downpoints of the communication-based systems; every object should be connected to the network in order to obtain a 100% accuracy [20, p. 1173].

2.2.2 Intelligent Transportation Systems

Intelligent Transportation Systems or *ITS* refer to systems that utilize computer control and communications for enhancing safety [21, p. 7]. The communications can be divided into two protocols: *Vehicle-to-vehicle* or *V2V*, where vehicles communicate directly to

each other, and *Vehicle-to-infrastructure* or *V2I*, where vehicles communicate with intelligent infrastructure surrounding the operational environment [21, p. 1123], [3, p. 4], [21, p. 198].

Both V2V and V2I depend heavily on wireless connections. IEEE 802.11p is a standard developed for ITS [21, p.23]. The standard is implemented for short-range connections. It utilizes licensed radio frequency 5.9 GHz [21, p. 44].

V2V-applications are based on multiple vehicles sharing their data with each other by means of wireless communication. The shared data may be their position, velocity, acceleration and even RADAR or other sensor data, forming a more complex fusion network [21, p. 198], [3, p. 8]. The obtained data can be used for adjusting speed or direction in order to avoid a collision.

V2I-applications are systems where vehicles share their data into the infrastructure, which in turn broadcasts it to other vehicles [21, p. 1123]. The infrastructure has multiple nodes handling the V2I communications, and these nodes share information also with each other, creating a large-scale network aware of the traffic situations in the area of each node.

One study proposes a GSM-based information system for railroad traffic [34]. The study presents a concept, where the trains communicate with a Train Traffic Centre, which is aware of each train's position. If two trains on the same track would be likely to collide, one of the trains would be stopped or its velocity would be decreased [34, p. 2]. Operationally this system resembles partially the Monorail transfer's current system.

2.2.3 Traffic applications for Machine Vision

Machine Vision is rarely the only perception system applied into traffic applications. Camera-based vision is usually used together with a bunch of other sensors, as in [37], [29] and [2].

In [2], Machine Vision is used for extracting distinct edges from the images captured of the environment. The edges are then connected to form a region, which is iteration by iteration reduced until it resembles a rectangle [2, p. 458]. The size of the final rectangle determines whether the object is another vehicle or something else. A RADAR is used to support the detections made by the camera and to provide depth information for forming 3D model of the environment [2, p. 459].

In [37], LiDAR and Machine Vision are used in fusion [37, p. 1154]. Machine Vision is used for extracting edges from the image data. This allows for tracking the scene [37, p. 1154].

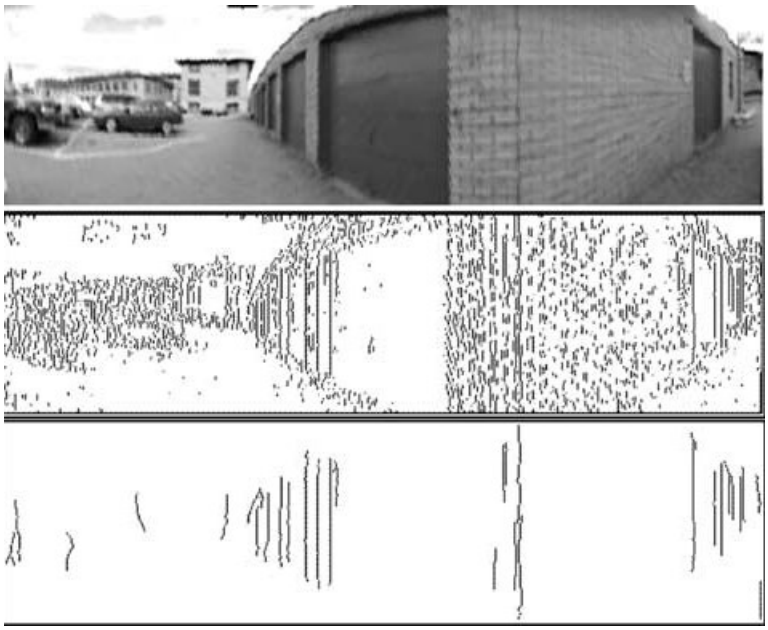


Figure 13: *Edge detection process [37, p. 1154].*

Figure 13 illustrates the process of extracting distinct edges from image data. Here the edges are detected around areas of different color while comparing to the surroundings.

Figure 14: *An example of Machine Vision-sensor-fusion [37, p. 1156].*

Figure 14 presents an example of the fusion data. The white circles represent camera-based detections, while the white stripe illustrates the laser scanner detecting the curb on the roadside [37, p. 1156]. As seen on the figure, the camera and the laser scanner are detecting different types of objects, replenishing the system's awareness of its surroundings.

2.3 Collision avoidance in robotics

According to the publicly released studies used as references for this work, the collision avoidance systems in industry are rarely entirely dependent on inter-device communications. They are most usually utilizing either wholly sensor-based systems or a fusion of sensors and communications instead, as in [31], where the robot provides obtained sensor data to a computer, which in turn makes decisions for avoiding collisions, and passes them back to the robot.

2.3.1 Sensor-based robotic applications

The most used sensors in the studied robotic collision avoidance systems are optical, ultrasound and electromagnetic sensors [8, p. 197], [13, p. 1], [7, p. 29]. LIDARs and LADARs are often utilized in mobile robots [22, p. 1], [23, p. 1669] instead of pointer-type infrared sensors for providing a wider FoV with a smaller amount of sensors.

The sensor data is utilized in multiple ways. One interesting approach to collision avoidance is the usage of *Virtual Force Fields* or *VFF*s (also called *Repulsive Virtual Force* [24, p. 3], *Potential Vector Fields* [25, p. 253]). The principle of the VFFs is the following:

- Objects are sensed with sensors
- The ultimate goal of the robot generates attracting force
- All other objects create repulsing force
- The sum of all forces is applied upon the robot virtually, leading to generation of new paths when necessary [26, p. 54], [31, p. 1181].

This kind of systems are useful especially for holonomic mobile robots.

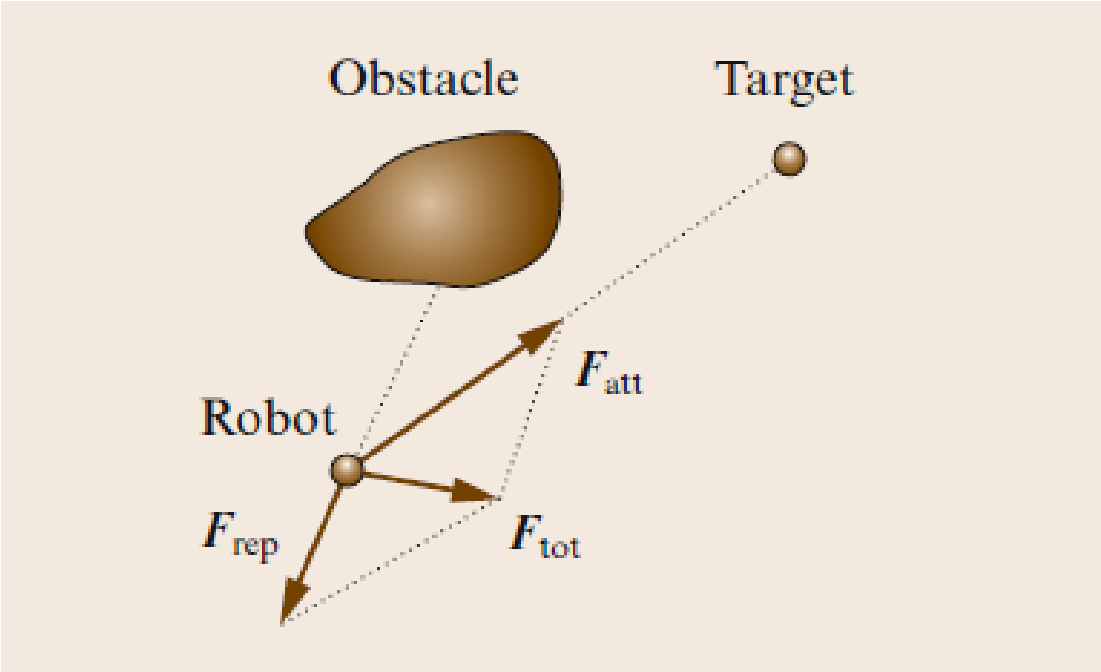


Figure 15: *An illustration of a VFF [6, p. 840].*

Figure 15 illustrates the basic principle of virtual force fields. The virtual force F_{tot}, which is the sum of the forces F_{rep} and F_{att}, causes the robot to avoid the obstacle and thus avoid collision.

Another intriguing approach is called *"Occupancy Grids"* [6, p. 855]. The method is based on dividing the operational environment into square-shaped grids. The presence of objects within these squares is detected with sensors. If an object is detected, the square is considered occupied and hence the area is avoided while planning a route [6, p. 855]. [38, p. 191-192].

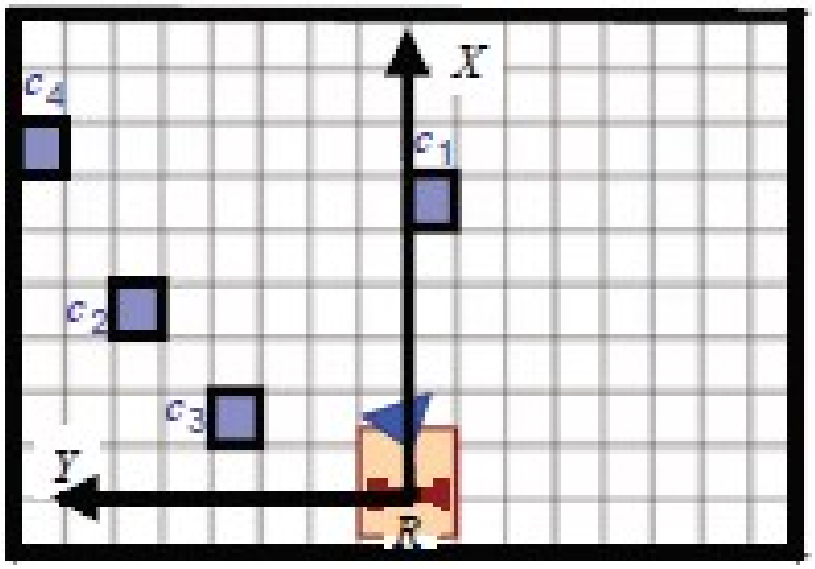


Figure 16: *An example of an occupancy grid [38, p. 191].*

Figure 16 illustrates the principle of occupancy grids. The squares marked with blue color are considered occupied and are avoided while planning possible future trajectories.

2.3.2 Machine Vision in robot applications

Machine vision can be implemented in robots as well as in vehicles. The related studies [35], [32] and [36] reveal that Machine Vision is rarely implemented as a standalone CAS for robotics, but instead it is used in fusion with other sensors, such as single-point laser distance sensors in [35, p. 275], ultrasonic sensors in [32, p. 526] and [36, p. 218].

Instead of collision avoidance alone machine vision is more often utilized also for the processing purposes as in [49]. It can be used for detecting and recognizing objects, their colors, poses and other features. Meanwhile it can still check for possible collisions.

2.3.3 Other types of collision avoidance

Other types of collision avoidance are usually implemented for stationary manipulators resembling human arms instead of mobile robots. One implementation for example measures electrical current flowing through servo motors moving the manipulator [20, p. 357]. If the current rises over a pre-defined threshold, the control system interprets this as a collision and stops the movement.

One approach is programming the robots in a manner that they will not collide with each other, as in [7, p. 32]. While this may be a robust system for avoiding inter-robotic collisions, it does not take collisions with other objects into account.

3. THE PROPOSED NEW IMPLEMENTATION FOR MONORAIL TRANSFER

The new implementation was planned carefully following the requirements defined with the help of internal Monorail documentation and various conversations with the Monorail Transfer Product Manager as well as other automation engineers. This chapter describes the process of preliminary planning, defining the requirements, choosing appropriate hardware and designing and implementing their installation and designing and implementing the software necessary for the CAS.

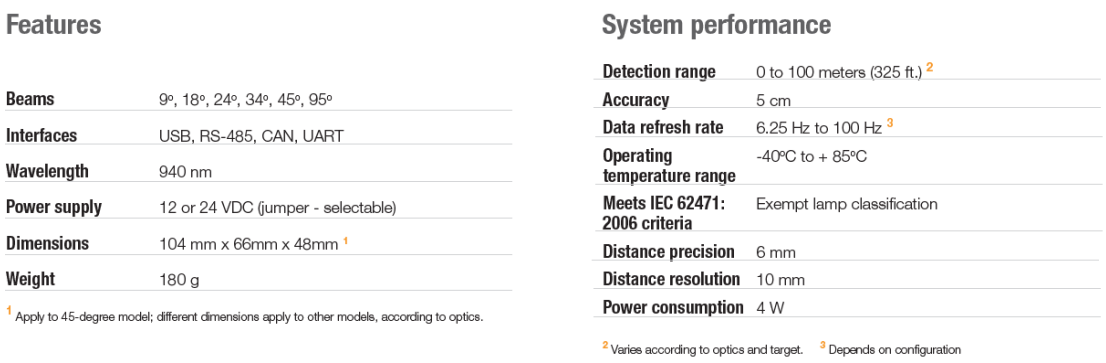
3.1 Preliminary planning

To get the work started it was necessary to perform some preliminary planning and choose some pieces of equipment. Preliminary planning was performed using the following information:

- The robot moves up to 5 m/s
- The braking distance is about 7 m
- Carriers in lines may travel in both directions
- Carriers in loops may travel only in one direction
- The system will not depend on Wi-Fi
- The system should be able to distinguish other carriers from other obstacles
- The system should be able to detect objects behind a curve of the rail
- The system should communicate with the PLC's via Ethernet/IP
- The system should detect other carriers from at least 10 meters
- The system should be able to observe also the sides and the underside of the carrier's path.

The Product Manager gave some tips about appropriate sensors for this application. Wide-range LED scanners were to be investigated. The first was a Pepperl-Fuchs R2100, which is a multi-ray 2D scanner. The R2100 can perform 2D measurements [40]. However the R2100 has a range limited to 8 m for objects with 95% reflectivity [40] and the diameter of the light spot is 0.5 m already from distance of 4 m.

The other proposed sensor was LEDDARTech M16, or more precisely, an evaluation kit consisting of a suppressed model of M16. The M16 is a multi-segment LED scanner with a maximum range of 100 m. The sensor has 16 segments which can be observed independently. The sensor provides distance and amplitude data for each segment.



Features

Beams	9º, 18º, 24º, 34º, 45º, 95º
Interfaces	USB, RS-485, CAN, UART
Wavelength	940 nm
Power supply	12 or 24 VDC (jumper - selectable)
Dimensions	104 mm x 66mm x 48mm [1]
Weight	180 g

[1] Apply to 45-degree model; different dimensions apply to other models, according to optics.

System performance

Detection range	0 to 100 meters (325 ft.) [2]
Accuracy	5 cm
Data refresh rate	6.25 Hz to 100 Hz [3]
Operating temperature range	-40ºC to + 85ºC
Meets IEC 62471: 2006 criteria	Exempt lamp classification
Distance precision	6 mm
Distance resolution	10 mm
Power consumption	4 W

[2] Varies according to optics and target. [3] Depends on configuration

Figure 17: *LEDDARTech M16 data sheet. [42]*

Figure 17 reveals other key features of the M16. The beam – the FoV – can be customized while ordering the sensor. The vertical FoV for the M16 is 7.5° independent of the horizontal FoV. M16 has an RS-485-interface and support for MODBUS protocol.

The segmentation of M16 inspired an idea of implementing an algorithm which reads the segments independently, and by combining the distance and amplitude data from the segments, it recognizes objects either as carriers or other obstacles. Initially the widest FoV available seemed applicable, but later drawings with Visio pointed out that the width of the beam is irrelevant after a certain threshold.

The Visio drawings revealed also that the M16 should be installed facing straight to the direction the carrier is traveling. This led to the situation where the M16 alone would not be enough for detecting objects behind the curves in the rail. Thus, some auxiliary sensors were introduced to the plan.

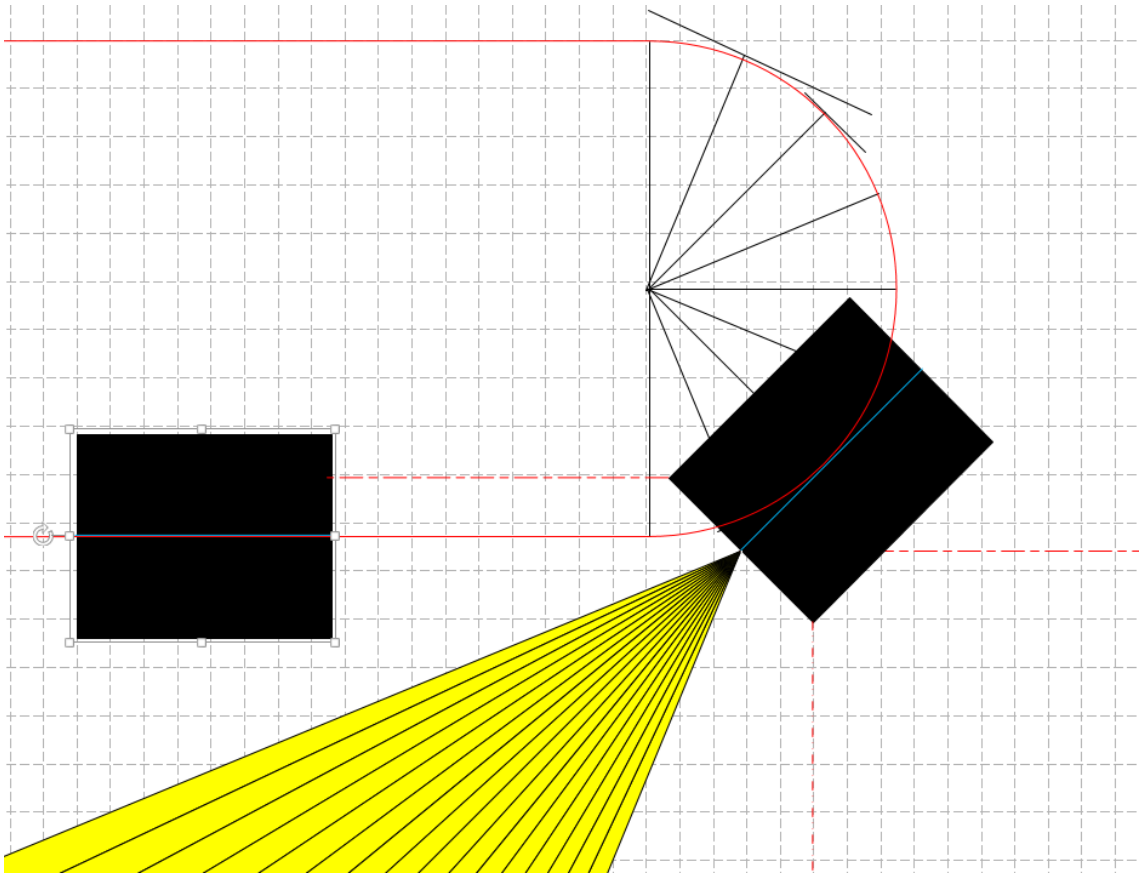


Figure 18: *Justification for the auxiliary sensors.*

Figure 18 illustrates the situation where the auxiliary sensors are useful. The black blocks represent Monorail carriers, the uniform red line is the rail, the yellow pattern is the M16's segmented beam and the red, dashed line between the carriers is the auxiliary sensor, installed onto the carrier equipped with M16. As there cannot be any solid structures on the actual path of the carrier, a passage with the width of the carrier is always unpopulated of the structures. Thus the auxiliary sensor can detect obstacles also around the curve.

The solid structures on the sides of the carrier's path inspired the idea of capturing the sensor data from the M16 with some pre-defined capturing frequency during a single round trip among the rail's whole length, having a rail clear of other carriers and other obstacles. Also the data from the absolute optical encoder would be captured, and the data points from the encoder and the M16 would be paired. Thus there would be data consisting of the position of the carrier and the normal detections in that position. The captured fusion data is called *"route chart"*. The route chart is then used for the collision avoidance within normal operation. The run-time sensor data is compared to the route chart, and if there are exceptions, certain operations are to be triggered.

The obstacle recognition algorithm mentioned before was planned to work as follows:

- The system detects an exception to the route chart
- The system checks which segments are responsible for detecting the exception

- The system checks the distances and their possible differences measured by the aforementioned segments
- The system checks the amplitudes and their possible differences from the aforementioned segments
- The system checks the amount of the exceptional segments
- The system checks whether the exceptional segments are either separate, adjacent or a combination of these.

According to these observations, the system decides whether the exception was caused by another carrier, some other obstacle or a combination of both.

3.2 Defining the Requirements

The following requirements were gathered using the internal Monorail documentation and having conversations with the Monorail Transfer Product Manager. Some of the requirements were drawn from the preliminary plans presented earlier.

- The system shall detect objects from a distance of 10 meters
- The system shall be able to detect objects around the corners of the rail
- The system shall be able to stop the Carrier if an object is detected
- The system shall be able to detect objects not only in the front-facing direction, but also beneath the Carrier's path
- The system shall detect other carriers and other objects
- The system shall not depend on Wi-Fi in any manner
- The communications from the System to the PLC shall be implemented using Ethernet/IP as the communication protocol
- The system shall provide fresh data for each cycle of the PLC (frequency 100 Hz)
- The system shall have enough storage capacity for storing the route chart into non-volatile memory
- The system shall have enough computing power for comparing the live data into the route chart in real time
- The system shall withstand ambient temperatures up to 55°C.

The requirements set some limitations for both the hardware and the software. They also provide a framework for designing and constructing the system.

3.3 Selecting the prototyping hardware

After an intense period of going through applicable devices from various vendors, the following pieces of equipment were selected for prototyping this implementation:

- Leddartech M16 LED scanner with a 24° FoV
- Ifm O5D100 optical proximity switch, Sn 2m
- Ifm O5D150 optical proximity switch, Sn 2m
- Raspberry Pi 2 Model B
- LinkSprite RS-485 shield for Raspberry Pi
- Murr MDD 5VDC switching power supply
- 48 kOhm resistors for lowering the sensor signal before entering the RasPi
- 120 Ohm terminator resistors for the RS-485-bus.

The M16 was chosen as it has a range which covers all the possible situations and 16 independent segments needed for the planned prototype. The optical switches were chosen according to their nominal sensing distance *Sn* and price.

Raspberry Pi or *RasPi* was chosen for pre-processing the data before entering the PLC. RasPi is not qualified for industrial environments but it provides a suitable platform for prototyping new applications at low cost. RasPi's maximum operation temperature is unknown, as it is not mentioned in its datasheet [Z]. It was anyhow chosen for the prototype for rapid implementation.

The RS-485 shield was needed for connecting the M16 and the RasPi without programming device-level logic into the RasPi's GPIO pins. The power supply was needed for the RasPi as there are no 5VDC outputs in the Monorail Carrier's onboard control cabinet.

Resistors were needed for two reasons; for lowering the signal level coming from the optical switches' outputs prior to introducing them to the RasPi and for bus end termination in the RS-485 bus.

3.3.1 Comparison between the microcontrollers

Throughout the development process the usage of a RasPi in an industrial environment faced doubts by multiple parties. This led to some additional research work for comparing some of the available microcontrollers to find the most suitable candidates for this application.

The contestants were measured in terms of CPU power, available memory, I/O-versatility, expandability, shock resistance, programmability, price and size. Since the Collision Avoidance System developed here is a prototype, many of the compared controllers are actually development boards.

The following table reveals the key features of each controller obtained from their data sheets and vendors.

Table 2: *Comparison between the microcontrollers [43], [44], [45], [46], [47], [48].*

	RasPi 2b	Intel Joule	Inico S1000	Arduino Uno	NI sbRIO-9651	Nexcom NISE 91
CPU: Cores/freq. [MHz]	4 / 900	4 / 1700	1 / 55	1 / 16	2 / 667	1 / 1000
RAM [GB]	1	4	0.008	2e-6	0.5	1
RS-485	Via adapter	N/A	N/A	Via adapter	2*on-board	On-board
I/O	GPIO	GPIO	8xDI/DO, 4xAI	GPIO	160 FPGA I/O	8*DI/DO
Expandability	GPIO	GPIO	Modular	GPIO	Modular	USB
Shock Resistance [g]	N/A	N/A	N/A	N/A	N/A	20G
Programmability	Any language	Any language	IEC 61131-3 ST	Arduino	NI LabView	Any language
Size [$mm_x mm_x mm$]	86*54*17	48*24*3,5	120*101*23	60*80*20	44*76*8	59*140*167
LAN ETH	1	0	1	No	1	2
Chassis Included	No	No	Yes	No	No	Yes
Price [€]	39	334	300	30	2490	430

In terms of processing power, Intel's Joule wins the comparison as Table 2 shows [43]. In terms of I/O versatility, NI sbRIO-9651 wins the game [47]. In terms of power vs. price, RasPi clears the table [48]. Nexcom's NISE 91, which is a compact, low-cost industrial PC, is the only product which had any data about vibration resistance [44]. The same device has also an on-board RS-485 interface, 2 LAN Ethernet ports and sufficient computing power, making it the only serious alternative to the RasPi.

3.4 Designing the software

The software running on the Raspi was implemented mostly with Node.js, utilizing a *Modbus-serial*-npm package. Java was also tried but the system would perform better with modular Node.js programming. C and C++ were also considered, but Node.js proved to be the most suitable candidate for this purpose due to the following facts:

- Node.js includes a ready-made serial communication API which allows for coding the telegrams bit-by-bit
- The program can be divided into modules which can be triggered when needed, sparing the scarce resources of the Raspi

- There are API's by Leddartech for C and C++ but they are poorly documented
- Node.js was the most familiar programming language.

3.5 Setting up the prototyping environment

The system was built up in parts since the equipment arrived in separate batches. First arrived the Raspi and its memory card. Raspbian "Jessie" was flashed into the SD card, system booted and a USB-WLAN-adapter installed. Next the programming environment containing of Java and Node.js was set up.

Next arrived the RS-485 adapter for the Raspi. It was fitted directly into the GPIO pins on the board. Then the Raspbian was modified to allow serial communication through the serial port. Console login via serial was disabled, UART was enabled.

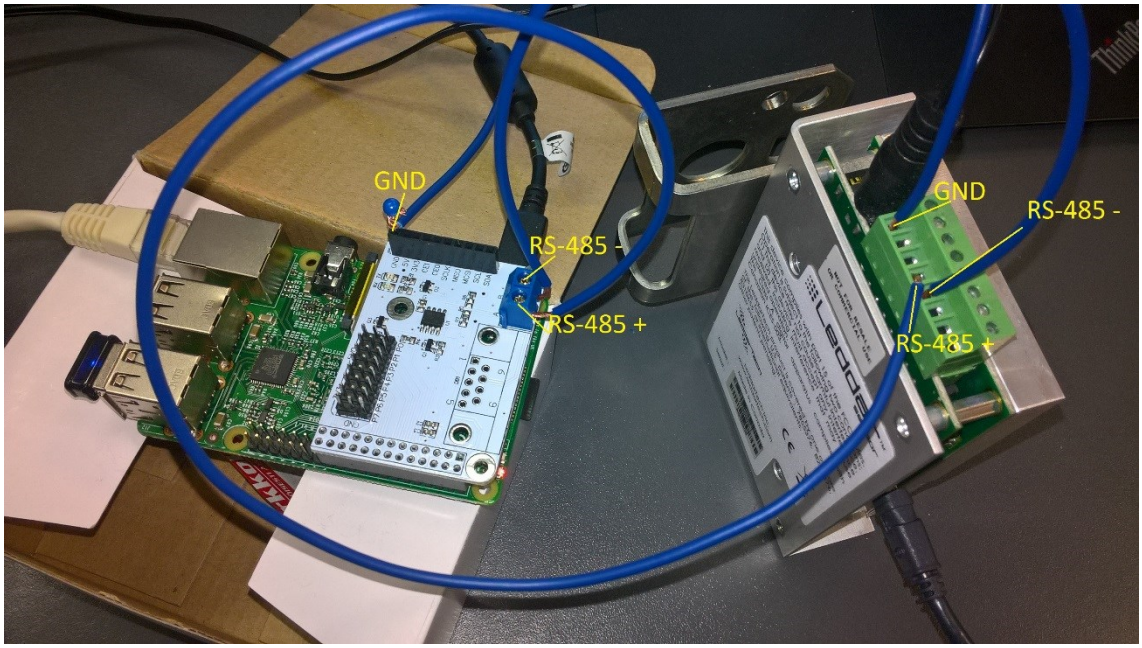


Figure 19: *Testing configuration.*

Figure 19 illustrates the configuration used in the following tests. On the right side the Evaluation Kit-version of M16 can be seen. On the left side there are the Raspi and the RS-485 adapter.

The system was now tested with the Evaluation Kit-version of Leddartech M16. First readings were now acquired with a Node.js program.

```
pi@raspberrypi:~/Desktop/JS $ node test2.js
[ 6, 5, 3, 3, 4, 4, 3, 3, 4, 1, 5, 5, 6, 5, 10, 9 ]
```

Figure 20: *First readings from the sensor.*

Figure 20 illustrates the obtained data. The readings are consistent, since the sensor was facing an object on the desk in its near proximity. Next the test was performed with the sensor lying approximately 0.3 meters from a wall.

```
pi@raspberrypi:~/Desktop/JS $ node test2.js
[ 37, 31, 29, 27, 29, 29, 29, 29, 29, 27, 31, 32, 32, 34, 41, 0 ]
```

Figure 21: *Readings from the second test.*

Figure 21 reveals the results of the second test. The far-right segment shows zero, because the sensor's maximum range was configured to 4 meters, and the beam escaped because the wall ended.

However, these tests only provided the data of a single detection. A quick look on the hardware setup revealed that the 120Ω resistors required for RS-485 were missing. The resistors are needed for damping signal reflections at the bus ends [19, p. 159]. After adding one of these in both ends of the bus and matching the length of the cables, the system started working properly.

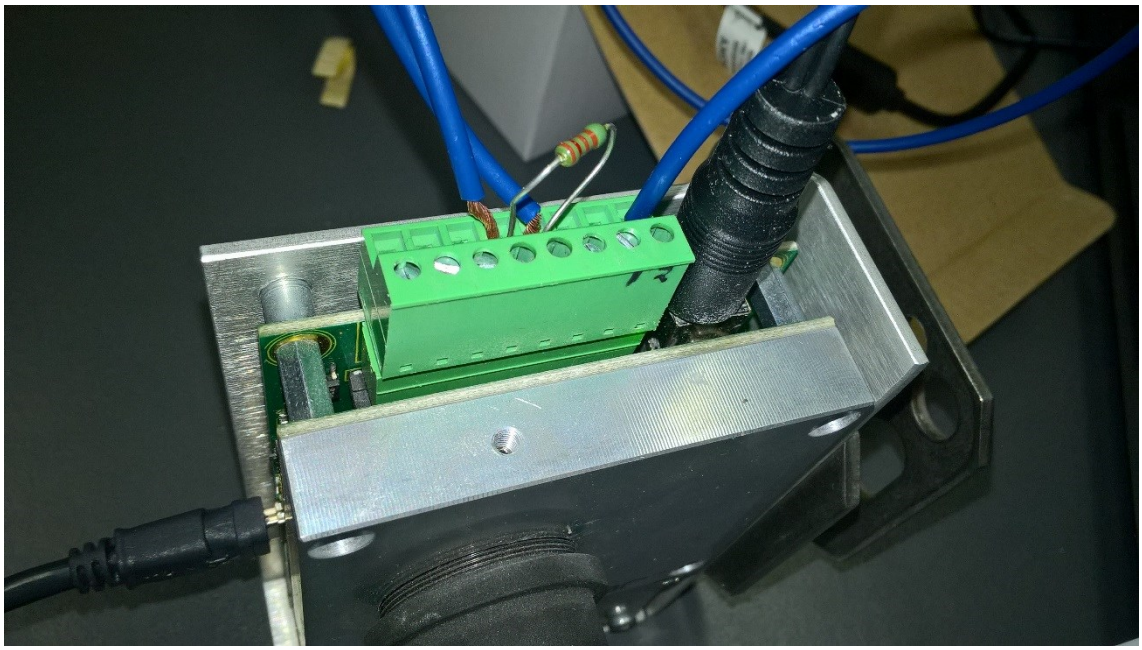


Figure 22: *120Ω resistor at the end of the bus.*

Figure 22 illustrates the resistor in the testing setup. The resistor was simply placed into the screw terminal between RS-485+ and RS-485-.

Now the system would allow to capture multiple frames sequentially. However the capture rate appeared quite modest. The rate was increased at the Node.js program utilizing an experimental npm package called *'nanotimer.js'*. Nanotimer.js allows to set timeouts and intervals in nanosecond-level while the standard node.js timer allows only for millisecond-level operations.

```
function seq(){
timer.setTimeout(FC04, '', '10m');
//console.log("Reading serial input");
}

timer.setInterval(seq,'','142u');
```

Figure 23: *Speeding up the capture.*

Figure 23 reveals how the increase in the capture rate was implemented programmatically. The timeout was first set to 10 milliseconds for obtaining a scanning rate of 100Hz. The interval between each scan cycle was experimentally iterated to 142 microseconds. There were attempts to calculate the correct interval time based on the baud rate and the structure of a standard Modbus-RTU-message, but the program would not work with the calculated value. The value was iterated a bit higher than the calculated value.

Next arrived the commercial version of the M16. The system was rebuilt in order to improve the connections. Terminal resistors were renewed, and the ground connection was implemented at the GPIO instead of the stack header and with a proper connection pin.

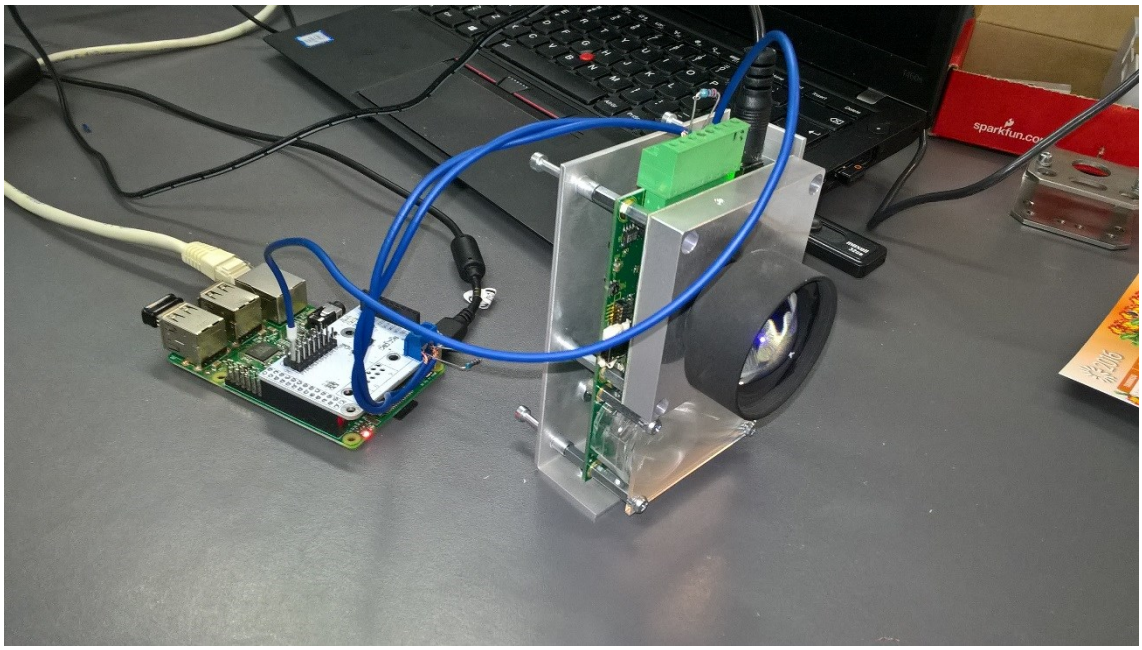


Figure 24: *Implementation with the commercial sensor.*

Figure 24 shows the re-built system. Now the baud rate was set to 230400 baud/s and the polling rate at the node.js code re-iterated to 6.061750 milliseconds. Now the system would allow scanning with a rate of 200 Hz, if neither the M16 nor the Raspberry Pi were loaded with other tasks simultaneously. This leaves a sufficient marginal, since the required minimum scanning rate was 100 Hz. The requirement was set by the fact that the PLC's in the Monorail's control cabinet are programmed with a 10ms scan cycle.

3.6 Implementing the software

After receiving the commercial version of M16 and setting up the environment, the actual programming work was started. The UML diagrams representing the software modules are presented in the appendices.

3.6.1 Reading the sensor data

The first implemented module was a program for observing the data in the console in real time. Basically this consisted of Modbus function code *FC04*, read input registers, which returned the sensor data to be displayed on the console.

```
//Implementing function code FC04: Read Input Registers.
//Reading 16 registers starting from 30016
function FC04(){

    client.readInputRegisters(16, 16, function(err, data) {
        if (data === undefined){
            //console.log("undefined");
            return;
            }
        console.log(data.data);


        var e1 = data.data;
        return(e1);
    });

}
```

Figure 25: *The core function for reading live sensor data.*

Figure 25 reveals the code snippet responsible for reading the sensor and visualizing the values. The *"readInputRegisters"* of class *"client"* was a ready-made method within the *"Modbus-serial"* npm package.

Since the baud rate and the polling frequency are high, the data stream contains pretty much errors even despite of the terminal resistors. The system was tested with multiple

types of wires and resistors, but the error rate was the same. The polling frequency and the baud rate were lowered, but this did not improve the situation at all. However, the effective data rate is much higher than the required rate, leading to the situation where the erroneous data can be discarded. This is attained by the command *"return"* if the gained data is *"undefined"*.

3.6.2 Saving the route chart

Next step was to save the data into non-volatile memory in order to compare it later to live sensor data. The captured detections form the *"route chart"*, which is used for neglecting detections occurring from objects belonging to the sides of the Monorail carrier's path.

```
var i = 0;
        var j = 120;
function FC04(){
    client.readInputRegisters(16, 16, function(err, data) {
        if (data === undefined){
            //In case of transmission error / CRC error/ error just
            //discarding the data and returning to scanning.
            return;
            }
        console.log("Iteration count: " + cnt);
        cnt++;


        // Desperate attempt to index the M16 data
        // with the position data from the WCS3B
        var pos = j;
        inputRegs[i] = (data.data);
        //console.log(inputRegs[i]);
        var file = './routechart.txt';


        var data2 = data.data + ',' + pos + ',' +  '\n';
        console.log(data2);
        //Writing the data in the same file, array after array
        //Arrays separated with a \n to reduce
        //the need for data processing in later phases
        fs.appendFile(file, data2, function (err) {
            if(err){}
        });
        j++;
        i++;
    });
}
```

Figure 26: *Saving the sensor data into non-volatile memory.*

The function for saving the data is presented in Figure 26. Besides the data from the Leddartech M16, some other data from the Monorail's control system is saved into the route chart, namely the absolute position of the carrier. In the snippet above the position is only "simulated" data coming from the loop. The data is saved into a .txt file, each row representing one scanning cycle. The rows are separated with space just to make them a bit more human-readable.

3.6.3 Comparing the sensor data to the route chart

Now that the data could be captured and saved, the next step was to implement the comparison. This was done by calculating the absolute difference between each value in a row.

```
var diff = Math.abs(M16[i]-data[i]);
```

Figure 27: *Core comparison between the live data and the route chart.*

Figure 27 shows the core of the comparison. The array *M16* contains the live data of a single scanning row, the array *data* contains the data of a single row in the route chart. The value of *i* was incremented to 16, since the Leddartech M16 has 16 segments, each providing one value. The value at 17 was discarded at this phase since it is reserved for the position data.

The amount of differing segments is saved into an array, as well as the indexes of the violating arrays. This information is used for recognizing the object.

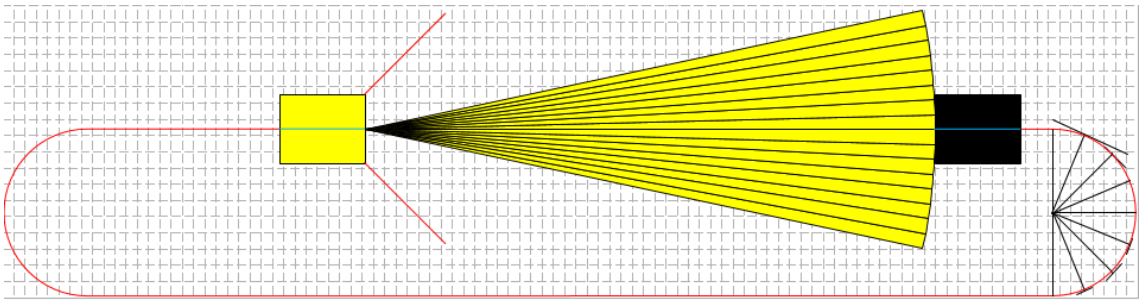


Figure 28: *The segmentation.*

Figure 28 clarifies the aforementioned process. In the picture the yellow and the black rectangular objects represent the Monorail carriers. The yellow one is equipped with the sensors. The distance between the carriers is 10 meters. Now that there's no curve in the next 10 meters and the segment violations occur in four segments equally distributed among the central line with similar distance data, it is safe to assume that the object is another carrier.

```
if (distToCurve > 10000 && (Math.abs(index1 - 5) > 1  || Math.abs(index4 - 10) > 1) && diffCount !== 0){
    // Now the detected object is most probably NOT another carrier
            myEmitter.emit('nCarrier');
            sensorDistance = meanValue_4(M16[index1-1],M16[index1],M16[index2],M16[index3], 4);
            console.log("Unidentified object detected at " + sensorDistance*10 + " millimeters");

}
//-----------------------------------------------------------------------------------------------
```

Figure 29: *Categorizing the detected object.*

Figure 29 shows an example from the code. The distance to the following curve is over 10 meters. The first violation arises from a segment which is closer to the central line as it would be if the object was a carrier. The fourth violation arises from a segment which is further to the right as it would be if the object was another carrier. Thus it is determined that the object cannot be another carrier. An event ‘nCarrier’ is emitted and the distance to the object is calculated using the mean value of the violator segments. The event ‘nCarrier’ triggers a function which sends a message to the PLC using nodepccc, telling it to set the target value of the velocity to 0.

```
var maxValue = Math.max(index1, index2, index3, index4);
var minValue = Math.min(index1, index2, index3, index4);
var sum = maxValue + minValue;

var variance = sum/2;

var diff1 = Math.abs(index1 - variance);
var diff2 = Math.abs(index2 - variance);
var diff3 = Math.abs(index3 - variance);
var diff4 = Math.abs(index4 - variance);

if (diff1 > 10 || diff2 > 10 || diff3 > 10 || diff4 > 10){
        console.log("The violations are not equally distributed depthwise, abort!");
        myEmitter.emit('nCarrier');
}
```

Figure 30: *Comparing the depth information of the segments.*

The values are compared also after detecting route chart violations, as shown in Fig. 30. When a violation is detected, the depth information of the violator segments is compared. The minimum and the maximum of the values are found, counted together and divided with the amount of the violations. Then the individual violator segments’ depth data is compared into the calculated “variance”.

Now that the data could be also compared, next step was to index the data according to the position sensor (Pepperl-Fuchs WCB3S, [40]). The position sensor is configured to update its position to the PLC in increments of 0.8mm. After consulting the Monorail Transfer Product Manager it was decided to index the data of M16 at every 8 mm of the WCS3B reading. The absolute accuracy of the M16 is ±5 cm [42], so 8mm steps are considerably low.

Now that the M16 data is saved at every 8mm, if the carrier is driving for example 800 mm/s, there will be 100 new data rows every second. The capture rate of the program responsible for the capture is functioning effectively at 150 Hz. Thus there will be redundancy of the M16 data. This should not appear as a problem, since the values with the same position index will not differ from each other significantly.

3.7 Linking the RasPi and the PLC

The connection between the PLC and the RasPi was implemented over Ethernet/IP using a node package called *'nodepccc'*. PCCC stands for *Programmable Controller Communication Commands*. The PLC's in the Monorail Carrier's Control Cabinet are provided by Rockwell. To be more precise, they belong to the Logicx5000- family in the Allen-Bradley line. The controllers don't support PCCC inherently; they are designed to function using the symbolic tag names. Thus the tags that were needed to be read or written needed to be mapped into PLC-5/SLC "files" before they could be accessed by external software.

3.7.1 Nodepccc

Nodepccc is an open-source library which is accessible via npm [50]. It allows using third-party software for accessing Rockwell/Allen-Bradley PLC's. Nodepccc does not support using symbolic tag names, so the tags in the PLC program need to be mapped into PLC-5/SLC file names before accessing them with nodepccc.

Nodepccc allows writing programs which can read and write the values in the PLC. In this implementation the values that are changed are the velocity of the carrier and the state of the collision avoidance system.

3.7.2 Modifying the PLC program

The Monorail Transfer is controlled by PLC's. The PLC's used in the prototyping version are manufactured by Rockwell, and they belong to the Allen-Bradley family. Programming happens using Studio5000 by Rockwell. Used languages are IEC-61131-3 compliant Ladder Diagram and Structured Text.

The PLC's used in Monorail Transfer prefer symbolic tag names instead of absolute addresses. For this reason, the tags that were needed to read or write were first mapped into PLC-5/SLC file numbers. After that they could be accessed with nodepccc.

The PLC program itself did not need much modification. There are tags for the sensor-based distance to the next carrier and for overriding the currently set target velocity. The

source for the sensor-based distance was changed from local inputs to the outputs of the Ethernet module, and the velocity override was programmed to replace its value if the set point from Raspi was lower than the current target velocity.

One new tag was introduced for sensing the state of the CAS. If the PLC is unable to initiate a connection with the Raspi within a certain time, the state is considered to be faulty. If the connection is initiated, the state is OK and the device is ready for operation.

3.8 Web-UI

There were wishes from the company that a web-based graphic user interface or *GUI* would be created for the collision avoidance system. One was made using node.js, HTML5 and client-side Javascript. The styles were created using CSS.

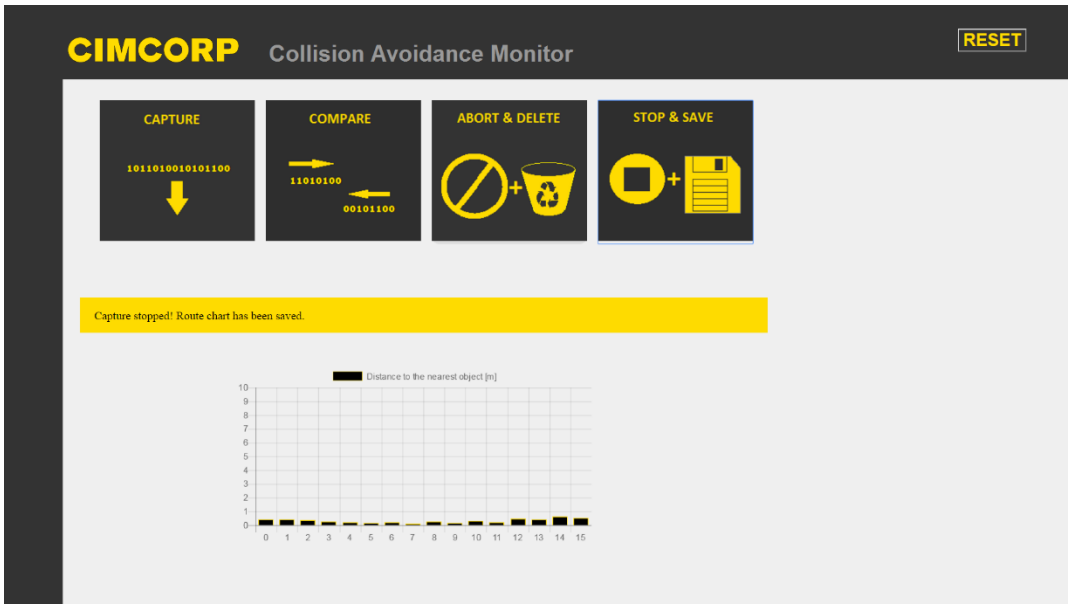


Figure 31: *GUI running on a browser.*

Figure 31 illustrates the GUI. There are buttons for initiating the capture of the route chart, starting the comparison between the route chart and live values and buttons for stopping the operation while either saving or discarding the data.

The bar chart illustrates the live values from the sensors. This was the core feature the Product Manager wanted to see in the GUI. The yellow area is for providing information to the user.

Connections between the GUI and the main software were implemented using socket.io and express.js. A server was programmed with node.js for serving the file and spawning the other modules as child processes, triggered by events caused by button pushes at the

GUI. This increases the modularity of the software, saving the scarce resources of the prototyping environment but increasing the latency.

4. VALIDATING THE PROTOTYPE

Unfortunately the system could not be tested to a massive extent in actual operational environment during the thesis work as the test loop at Cimcorp testing facility was reserved for testing some other equipment. The communications between the Raspi and the PLC were verified online and the passage of the commands throughout the system from web-UI to the PLC's was tested once. Position data from a running Monorail Carrier's PLC was captured with a node.js program for using it in offline simulations.

The functionality of the object recognition was tested without live connection to the PLC's. However the connection is known to work thanks to the few online tests. The core of the object recognition lies however in the utilization of the independent segments of the M16. This was tested extensively without the PLC-communications.

4.1 Capturing live values

Capturing the live values without saving them was the first implemented module. This was tested in an office environment having the sensor face a wall.

```
pi@raspberrypi:~/Desktop/JS/vanhat kehitysversiot $ node routechartcapture.js
[ 67, 65, 64, 55, 52, 40, 50, 34, 51, 38, 60, 48, 74, 69, 82, 78 ]
[ 67, 65, 64, 54, 52, 40, 51, 34, 51, 38, 60, 47, 74, 69, 82, 78 ]
[ 67, 65, 64, 54, 52, 40, 51, 34, 51, 38, 60, 48, 74, 69, 82, 78 ]
[ 68, 65, 64, 54, 51, 39, 51, 35, 51, 37, 61, 48, 74, 68, 82, 78 ]
[ 68, 66, 64, 55, 52, 40, 51, 34, 51, 38, 60, 49, 75, 70, 83, 79 ]
[ 68, 66, 64, 54, 52, 40, 51, 35, 51, 38, 60, 48, 74, 69, 82, 78 ]
[ 68, 65, 64, 54, 52, 40, 50, 34, 52, 39, 59, 47, 75, 69, 83, 78 ]
[ 68, 65, 64, 54, 51, 40, 50, 34, 51, 38, 60, 48, 74, 69, 82, 79 ]
[ 68, 66, 64, 54, 52, 41, 51, 34, 51, 38, 60, 49, 74, 68, 83, 78 ]
[ 68, 65, 64, 54, 53, 40, 51, 34, 51, 39, 61, 48, 74, 69, 83, 78 ]
[ 67, 65, 64, 54, 52, 40, 51, 34, 51, 38, 60, 48, 75, 69, 83, 79 ]
[ 68, 65, 64, 55, 53, 40, 51, 34, 51, 39, 60, 49, 74, 69, 83, 78 ]
[ 68, 66, 64, 55, 52, 41, 51, 34, 51, 38, 61, 49, 74, 69, 82, 78 ]
[ 67, 66, 64, 55, 52, 41, 51, 35, 51, 38, 60, 48, 74, 69, 82, 78 ]
```

Figure 32: *Testing the live capture.*

Figure 32 shows the console during the live capture. The values are consistent, as the sensor was approximately 0.7 meters from the wall and there were some items on the way which are visible as the lower values in the middle segments. This verifies that the sensor picks up reasonable values and the RasPi can interpret the Modbus messages sent by the sensor.

Next an item was placed approximately 0.2 meters from the sensor.

```
pi@raspberrypi:~/Desktop/JS/vanhat kehitysversiot $ node routechartcapture.js
[ 17, 18, 15, 8, 5, 0, 3, 65534, 3, 65535, 8, 2, 20, 12, 35, 19 ]
[ 17, 18, 14, 8, 5, 0, 3, 65534, 3, 65535, 8, 2, 19, 13, 34, 18 ]
[ 17, 18, 15, 8, 6, 1, 3, 65534, 3, 65535, 8, 2, 20, 12, 35, 19 ]
[ 17, 17, 15, 9, 5, 0, 4, 65534, 4, 0, 7, 1, 20, 13, 34, 18 ]
[ 18, 18, 16, 9, 5, 1, 4, 65534, 3, 65535, 8, 2, 19, 12, 34, 18 ]
[ 17, 18, 15, 8, 6, 0, 4, 65534, 3, 65535, 7, 2, 20, 12, 34, 18 ]
[ 17, 18, 14, 8, 5, 0, 3, 65533, 2, 65535, 8, 1, 20, 13, 34, 18 ]
[ 17, 18, 14, 8, 5, 0, 3, 65534, 3, 65535, 8, 2, 20, 12, 34, 18 ]
[ 18, 18, 15, 8, 5, 0, 4, 65533, 3, 0, 8, 2, 19, 13, 35, 19 ]
[ 18, 18, 15, 8, 5, 1, 4, 65534, 4, 0, 8, 1, 21, 12, 35, 18 ]
[ 17, 18, 15, 8, 6, 1, 4, 65534, 3, 65535, 8, 2, 19, 12, 33, 19 ]
[ 17, 18, 15, 8, 6, 0, 4, 65534, 3, 65535, 8, 2, 21, 13, 35, 20 ]
```

Figure 33: *Close-up capture.*

As Fig. 33 shows, the values become inconsistent if the object gets too close. Some of the values are 0 while some are at the 16-bit integer's maximum (65535). The object causing these values was red. The test was re-performed using a white object of similar size.

```
pi@raspberrypi:~/Desktop/JS/vanhat kehitysversiot $ node routechartcapture.js
[ 32, 33, 29, 21, 17, 10, 15, 6, 15, 8, 21, 13, 39, 27, 51, 38 ]
[ 32, 31, 29, 20, 18, 11, 16, 6, 15, 9, 21, 13, 39, 27, 51, 38 ]
[ 32, 32, 29, 21, 17, 11, 16, 6, 15, 9, 21, 13, 38, 27, 51, 39 ]
[ 33, 32, 29, 20, 18, 11, 16, 6, 16, 9, 20, 12, 39, 28, 52, 39 ]
[ 32, 32, 30, 20, 17, 10, 16, 7, 16, 9, 20, 12, 39, 28, 51, 38 ]
[ 31, 32, 29, 21, 17, 11, 16, 7, 15, 8, 21, 13, 40, 27, 53, 39 ]
[ 32, 33, 29, 20, 17, 11, 15, 6, 16, 9, 20, 12, 39, 28, 52, 39 ]
[ 32, 32, 29, 20, 17, 11, 16, 6, 16, 9, 21, 12, 38, 29, 52, 37 ]
[ 33, 32, 29, 20, 18, 10, 16, 7, 15, 8, 21, 13, 38, 27, 51, 38 ]
[ 33, 32, 29, 20, 18, 11, 15, 6, 14, 9, 22, 13, 38, 26, 52, 37 ]
```

Figure 34: *Testing with a white object.*

Now the values are far more consistent, as seen in Fig. 34. The color of the objects seem to interfere with the reflection of the infrared light. The results of this test suggest that the object to be detected should not be red. The results also give an idea of an appropriate threshold for considering the detections being at same distance if the object is close to the sensor.

4.2 Saving the route chart

After the data could be read live, the next logical step was starting to save it. This was tested using the same white object in the sensor's near proximity:

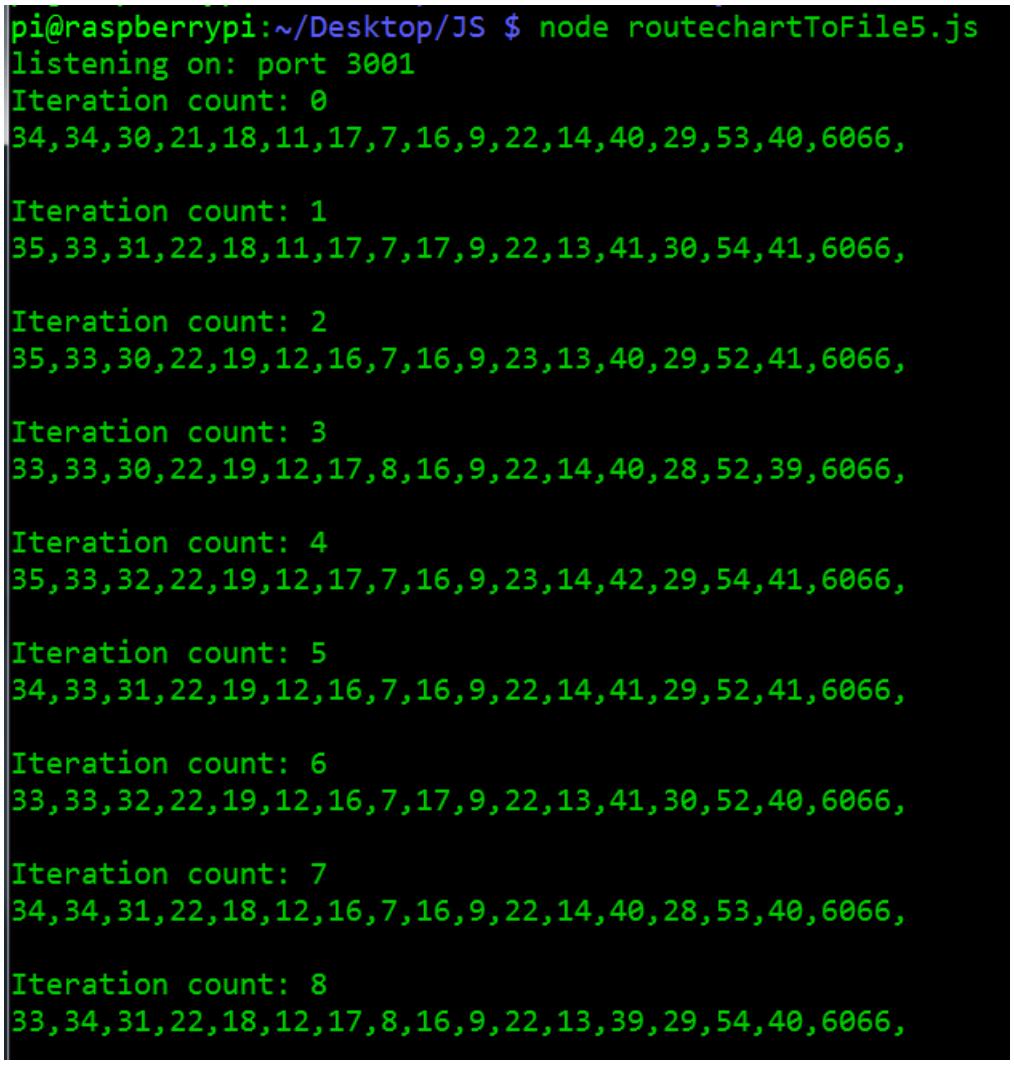


Figure 35: *Saving the route chart.*

Fig. 35 shows the console during the saving process. The “iteration count” is used at the prototyping phase for getting an approximation of the scanning frequency. The value “6066” at the index 17 is derived from a file, which includes the position values gained through RasPi-PLC-communications.

Corresponding values can now be found from a file routechart.txt:

```
34,34,30,21,18,11,17,7,16,9,22,14,40,29,53,40,6066,
35,33,31,22,18,11,17,7,17,9,22,13,41,30,54,41,6066,
35,33,30,22,19,12,16,7,16,9,23,13,40,29,52,41,6066,
33,33,30,22,19,12,17,8,16,9,22,14,40,28,52,39,6066,
35,33,32,22,19,12,17,7,16,9,23,14,42,29,54,41,6066,
34,33,31,22,19,12,16,7,16,9,22,14,41,29,52,41,6066,
33,33,32,22,19,12,16,7,17,9,22,13,41,30,52,40,6066,
34,34,31,22,18,12,16,7,16,9,22,14,40,28,53,40,6066,
33,34,31,22,18,12,17,8,16,9,22,13,39,29,54,40,6066,
34,34,31,21,18,12,17,8,16,9,23,14,40,28,52,39,6066,
34,33,30,21,19,13,17,7,16,9,22,13,39,28,53,41,6066,
34,34,31,22,18,12,16,7,17,10,21,13,40,29,53,39,6066,
33,33,32,22,18,12,17,7,16,9,22,14,39,28,52,40,6066,
34,34,31,22,18,12,17,8,16,9,22,14,41,30,52,42,6066,
34,34,30,21,18,11,16,7,16,9,22,13,41,30,53,41,6066,
34,34,31,21,19,12,16,7,17,10,21,13,40,30,52,40,6066,
```

Figure 36: *routechart.txt values.*

Fig. 36 verifies that the system is capable of saving the sensor values and the position data into non-volatile memory for later usage.

4.3 Comparing live data to the route chart

The comparison was tested by first saving the route chart and then introducing an object to the sensor's FoV. The result of the comparison depends on the current (simulated) position of the carrier, the amount of the violator segments, the distance of the violations and the adjacency of the detections.

```
Read count: 740
4838
Route chart violation @ M16/Array 0/0
M16[0]: 23 Array[0]:
34
Route chart violation @ M16/Array 2/2
M16[2]: 20 Array[2]: 30
Route chart violation @ M16/Array 10/10
M16[10]: 11 Array[10]: 22
Route chart violation @ M16/Array 12/12
M16[12]: 23 Array[12]: 38
Route chart violation @ M16/Array 13/13
M16[13]: 15 Array[13]: 27
Route chart violation @ M16/Array 14/14
M16[14]: 34 Array[14]: 52
Route chart violation @ M16/Array 15/15
M16[15]: 17 Array[15]: 37
DIFFCOUNT = 7
[ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 3, 4, 5, 6 ]
Scattered detections!
[3700,467254288 UNDEF] Preparing to WRITE L2:1
```

Figure 37: *Comparing the route chart to live data.*

Figure 37 shows the console output of such testing. At scanning cycle 740 a foreign object was introduced to the sensor. The distance to the nearest curve (simulated) is 4838 millimeters. There are violations on segments 0,2,10,12,13,14 and 15, indicating that there are multiple objects among the way since the violations are not adjacent. Thus the detection is considered scattered and the PLC's target velocity is overridden with 0. The distance which triggers a violation is set to 100 mm.

4.4 Reacting to violations

This chapter provides the test results from the tests concerning the correct reactions to certain types of violations. Acting as another carrier was particularly hard to reproduce in an office environment, but fortunately one occasion was captured successfully.

4.4.1 Reacting to another carriers

The system is expected to send a slow-down command to the PLC when another carrier is detected within a 10 meters range. This was simulated using a little piece of cardboard which was then introduced to the sensor during comparison between the route chart and the live data.

```
pi@raspberrypi:~/Desktop/JS $ node rcc9.js
Read count: 0
16434
Route chart violation @ M16/Array 6/6
M16[6]: 149 Array[6]: 162
Route chart violation @ M16/Array 7/7
M16[7]: 147 Array[7]: 164
Route chart violation @ M16/Array 8/8
M16[8]: 143 Array[8]: 162
Route chart violation @ M16/Array 9/9
M16[9]: 143 Array[9]: 167
Route chart violation @ M16/Array 10/10
M16[10]: 142 Array[10]: 165
Route chart violation @ M16/Array 11/11
M16[11]: 148 Array[11]: 170
Route chart violation @ M16/Array 12/12
M16[12]: 150 Array[12]: 163
DIFFCOUNT = 7
[ 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 0, 0, 0 ]
147 143 143 142 4
Preceeding carrier detected at 143.75 millimeters
Braking with full power
```

Figure 38: *Detecting another carrier.*

Figure 38 illustrates one of the rare occurrences when this action was simulated properly. The snippet is from an older version of the code, so the PLC communications are not yet present in the picture. They were however added to the code afterwards.

The carrier is driving a straight sector on the rail. The next curve is coming in 16434 millimeters. The system detects an object which begins from segment with index 6 and ends at segment 12. There are quite broad margins for the object to be accepted as another carrier widthwise. However it is important to notice that all the violating detections arise from distances ranging 142-150 cm, as shown in figure 37. The threshold for considering the distance similar is set to 10 cm, so now it can be considered that the detections arise from similar distances and the detected object is most likely another carrier. Because the distance to the carrier is so low, the system will now brake with full power. If the carrier was for example at 9 meters, the braking would be smoother.

4.4.2 Reacting to non-carrier objects

Reactions to non-carrier objects were easier to simulate without the actual operational environment. Introducing an item to the sensor during the comparison between the route chart and live data was usually enough for triggering this event.

```
Read count: 75
16434
Route chart violation @ M16/Array 12/12
M16[12]: 29 Array[12]: 39
Route chart violation @ M16/Array 13/13
M16[13]: 19 Array[13]: 29
Route chart violation @ M16/Array 14/14
M16[14]: 40 Array[14]: 51
Route chart violation @ M16/Array 15/15
M16[15]: 23 Array[15]: 37
DIFFCOUNT = 4
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3 ]
The object cannot be another carrier.
[15334,708572128 UNDEF] Preparing to WRITE L2:1
```

Figure 39: *Reacting to a non-carrier object.*

Figure 39 shows the console output in such situation. Now the sensor was covered with hand during scanning cycle 75. The distance to the next curve is 16434 millimeters. Now there are four violations arising symmetrically from the right-side segments. The detection cannot arise from another carrier, because the following 16.4 meters are only straight rail. The system sends a message to the PLC telling it to set the target velocity of the carrier to 0.

4.4.3 Reacting to scattered detections

It the detections arise from non-adjacent segments, there are most likely multiple objects among the carrier's way. This means that there are multiple non-carrier objects, a carrier and a non-carrier object or an object with bizarre morphology. However the carrier needs to be stopped in such situation.

```
Read count: 75
16434
Route chart violation @ M16/Array 0/0
M16[0]: 22 Array[0]:
33
Route chart violation @ M16/Array 2/2
M16[2]: 19 Array[2]: 30
Route chart violation @ M16/Array 6/6
M16[6]: 6 Array[6]: 16
DIFFCOUNT = 3
[ 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
Scattered detections!
[15998,367806651 UNDEF] Preparing to WRITE L2:1
```

Figure 40: *Scattered detections.*

Fig. 40 shows the console output in the case of scattered detections. The violator indexes are 0, 2 and 6. Once again, the PLC is commanded to stop the motion.

4.5 Running the system via the Web-UI

The Web-based GUI was also tested thoroughly for finding possible flaws and making sure that it has all the necessary functionalities. The capture, compare and both stopping options were initiated via the GUI and the flow of data to the PLC was verified.

For example detecting an object during a GUI-initiated comparison was tested:

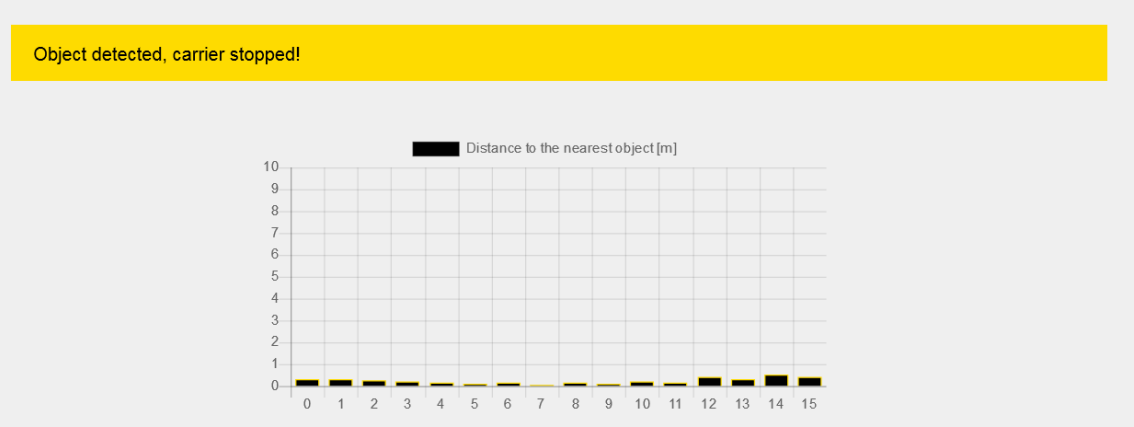


Figure 41: *GUI prompting the user about a detection.*

Figure 41 shows a screen capture from the GUI after a route chart violation. The yellow bar contains the text that is shown to the user.

During the testing some modifications were done. The modifications are presented in more detail later, but some of those are just illustrated in here. For example, guards for initiating inappropriate operations in certain situations were created and tested. Following image shows the GUI after trying to begin a comparison during an active route chart capture process:

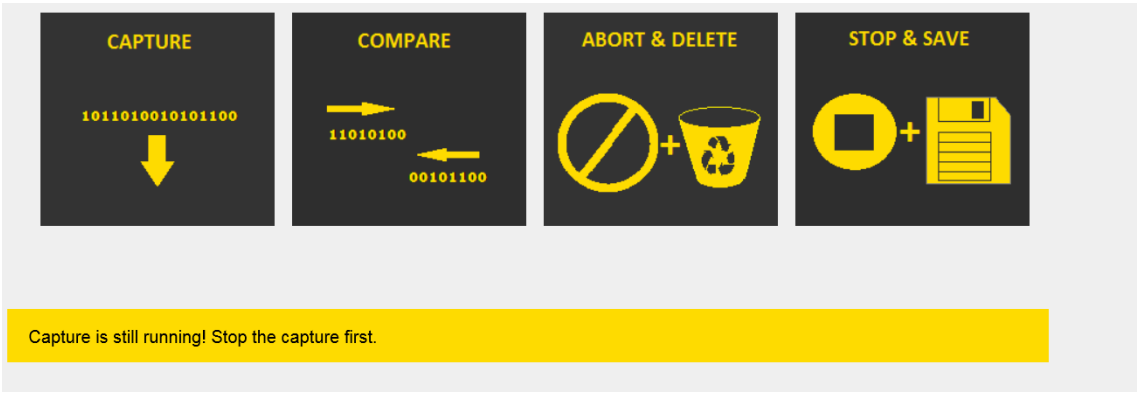


Figure 42: *Button guards in operation.*

Figure 42 shows the prompt that will be visible to the user after trying to initiate comparison before completing the capture successfully.

Other guards were tested and found functioning as well.

The GUI was tested with all the most common browsers in Windows environment (Google Chrome, Mozilla Firefox and Internet Explorer). There were no functional differences between the browsers, only some minor differences concerning the GUI's appearance. For example in Internet Explorer the buttons had round corners as they were originally written to be just original HTML buttons with round corners, not images. In other browsers the image-decorated buttons appear with sharp corners.

The yellow notification bar was also tested to be functioning by introducing objects to the sensor while running the CAS:

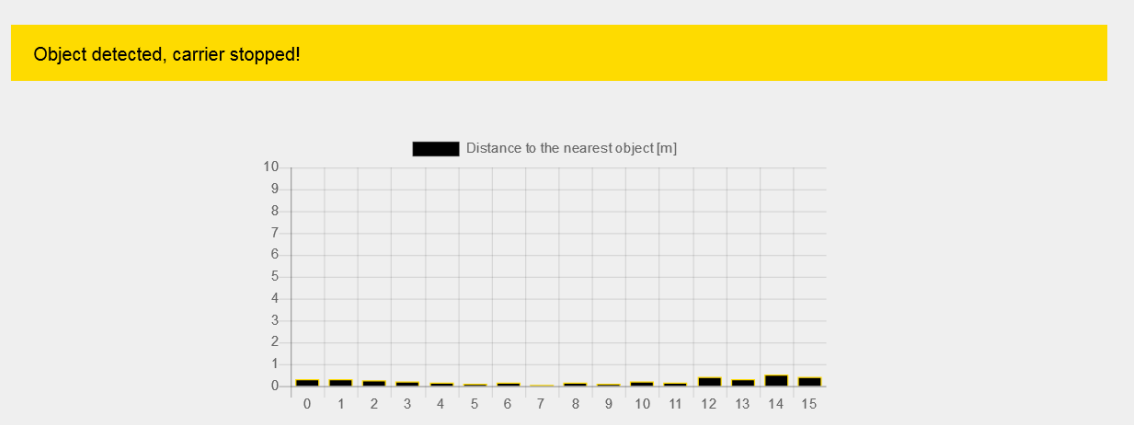


Figure 43: *GUI notifying the user about a detection.*

Figure 43 illustrates the GUI after the CAS has detected a violation to the route chart. The bar chart illustrates the distance data of the segments.

4.6 Modifications after testing

The node modules did not need any alteration as they were tested thoroughly during the development process. The GUI, on the other hand, was re-iterated multiple times during the testing process as it lacked some of the needed functions. The following bulleted list contains the added functionality:

- Guards for launching operations in inappropriate situations, for example trying to run the collision avoidance without having a route chart in the non-volatile memory
- Info labels for buttons when hovering with a mouse cursor
- A color scheme matching with the company's standards
- Better icons for the buttons
- Help button.

The guards for the operations were implemented with Javascript. For example pushing the button "Capture" first and "Stop & Save" after that will set a Boolean operator's value to true. When pushing the button "Compare", the code checks the value of the operator. If the value is true, compare can be activated. If it's false, compare cannot be initiated and the user is prompted about the situation. The operator's value will also be true if a pre-captured route chart can be found.

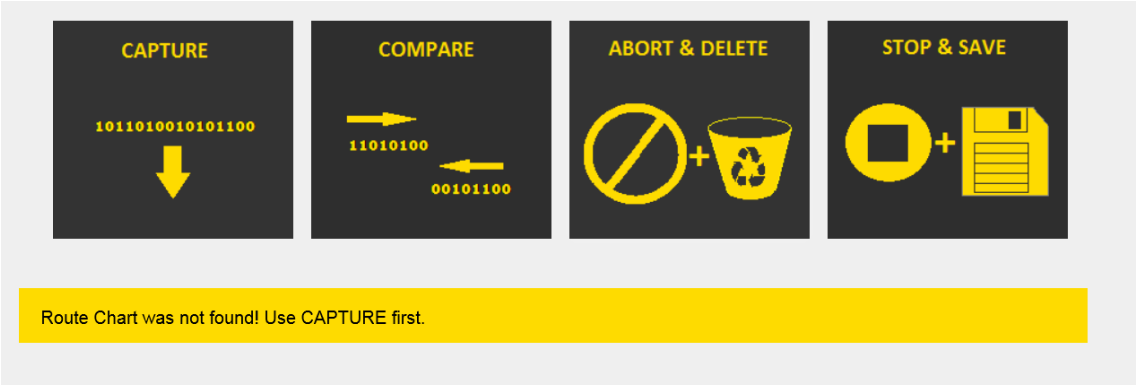


Figure 44: *Button guards in operation.*

Figure 44 illustrates one the aforementioned guard operations. “Compare” was pressed prior to capturing the route chart, and no pre-existing route chart could not be found in the non-volatile memory.

Other guards were implemented for pressing compare while capture is still active and vice versa. A help function was implemented and a button for that added.

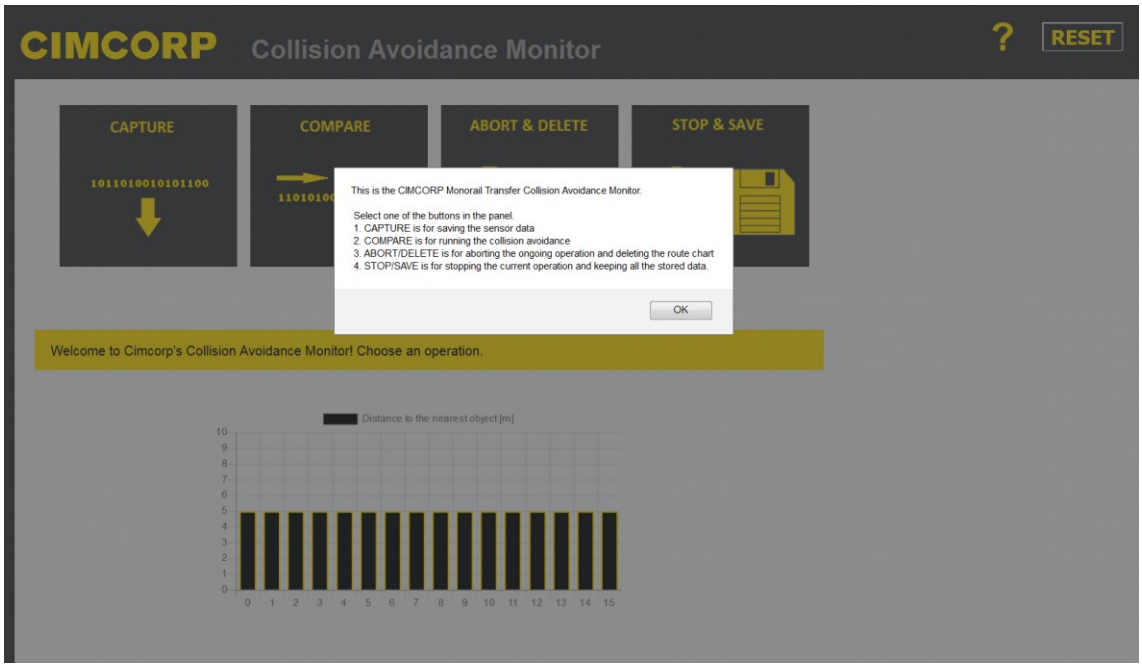


Figure 45: *The Help function.*

Figure 45 shows the main view after pressing the yellow question mark in the upper-right corner. The alert box gives the user basic information about the usage of the monitor.

One physical modification was also done; the RasPi and the RS-485 adapter were enclosed into a plastic housing which can be attached to a DIN rail.

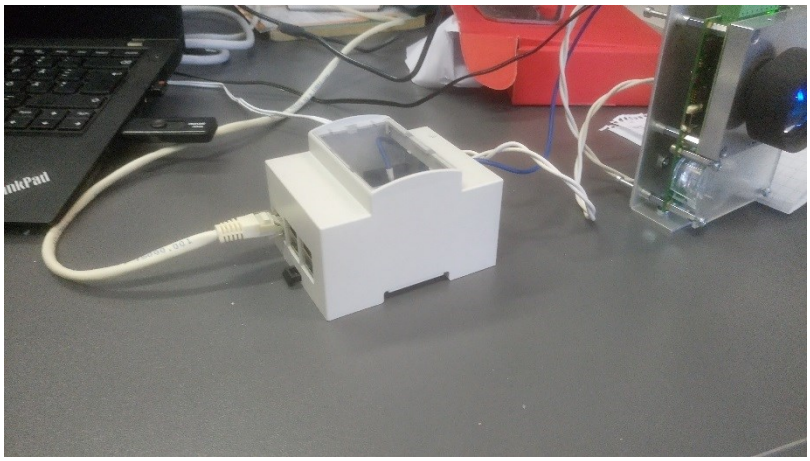


Figure 46: *RasPi in a chassis.*

Figure 46 illustrates the chassis with RasPi inside. The chassis needed also minor modifications as the RS-485 adapter's outputs were not accessible from the opening on the top of the chassis. A small hole was crafted into the side of the chassis for allowing the passage of the wires. Now the RasPi can be attached into the DIN rail in the Monorail Carrier's onboard Control Cabinet.

4.7 Findings

The performed tests revealed the characteristics of the implemented prototype. The response time for violations could not be even measured as they were perceived virtually immediately. The system reacts to detections correctly according to the carrier's own position. The system detects objects both near and far. The system is able to communicate with the PLC in both directions.

One restriction was found; the object to be detected cannot be red as the sensor will return non-consistent values after recognizing red objects. However this should not be a problem since the Monorail Carrier does not have any red parts which would be visible outside.

The GUI was found to be surprisingly resource-demanding during operation. The update rate of the visualization was initially set maybe a bit high.

5. CONCLUSIONS AND FUTURE WORK

This thesis introduced designing and implementing a functional prototype for an autonomous, less Wi-Fi-dependent Collision Avoidance System for Cimcorp Monorail Transfer. This work also introduced the sensor technologies used in existing collision avoidance systems and other technologies behind the systems.

5.1 Conclusions

Collision Avoidance Systems can be implemented using either sensor-based technologies, communication-based technologies or any combination of these. The technology must be chosen individually for each implementation, for the operational environments and other requirements may vary significantly between implementations. Sensor fusion increases the reliability of a Collision Avoidance System.

For the Cimcorp Monorail Transfer a sensor-based CAS was the optimal choice, since the Wi-Fi connection has better usage. Optical sensors were chosen for their lower price compared to electromagnetic sensors. The implemented prototype proved the suitability of the Leddartech M16 as the main sensor for the CAS and the functionality of the implemented software. If the implementation was categorized, it would perhaps be a fusion of a sensor-based system and machine vision, since the system utilizes optical data for recognizing the objects it encounters.

The strengths of the CAS are the multi-segment FoV, the precise algorithm which reacts to detections according to the carrier's own position, the user-friendly GUI and the fast response times. The weakest part of the system is the RasPi, as its reliability in industrial environments is unknown.

Overall the system meets all the requirements set before except for the ambient temperature. However, the implemented system is only a prototype which will never enter actual tire-manufacturing environment before some extensive refinement. Thus, the work can be regarded successful.

5.2 Future work

Possible future work could include replacing the RasPi with some industrial-grade controller, such as the Nexcom Nise 91 presented before. The M16 could be covered with a chassis or it could be changed to a similar product from the same manufacturer with an included chassis. The system should also be tested in the Monorail loop extensively for finding possible flaws that went undetected in the tests performed in office environment.

The GUI could be implemented for example in Java as a standalone application instead of an interactive HTML page.

REFERENCES

[1] T. Sheorey, N. Shrivas, Development of Sensor Based Front End Collision Avoidance System for Highways, Proceeding of the 2015 IEEE International Conference on Information and Automation, Lijiang, China, August 2015

[2] N. Srinivasa, Y. Chen, C. Daniell, A fusion system for real-time forward collision warning in automobiles, Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE, Oct.2003, pp. 457-463

[3] R. Adla, N.Al-Holou, M.Murad, Y.A.Bazzi, Automotive collision avoidance methodologies Sensor-based and ITS-based, Computer Systems and Applications (AICCSA) 2013 ACS International Conference on Computer Systems and Applications, 2013, pp. 1-8.

[4] J. Fraden, Handbook of Modern Sensors: Physics, Designs and Applications, Fifth edition, ISBN 978-3-319-19303-8, 2015, 765 pages.

[5] J. Wilson, Sensor Technology Handbook, ISBN 0-7506-7729-5, 2005, 589 pages.

[6] B. Siciliano, O. Khatib, Handbook of Robotics, ISBN 978-3-540-23957-4, 2008, 1611 pages.

[7] P. Brunn, Robot Collision Avoidance, Industrial Robot, Vol. 23, Number 1, 1996, pp. 27-33

[8] J. Graham, J. Meagher, S. Derby, A safety and Collision Avoidance System for Industrial Robots, IEEE TRANSACTIONS ON INDUSTRY APPLICATIONS, VOL. IA-22, NO. 1, JANUARY/FEBRUARY 1986, pp. 195-204

[9] A.Y.C. Nee, Handbook of Manufacturing Engineering and Technology, ISBN 978-1-4471-4669-8, 2015, 3485 pages.

[10] T. Kurfess, Robotics and Automation Handbook, ISBN 978-0-8493-1804-7, 2004, 554 pages.

[11] A.Discant, A.Rogozan, C.Rusu, A.Bensrhair, Sensors for Obstacle Detection - A Survey, 30th ISSE, 2007

[12] M. Skolnik, Radar Handbook, 3rd. edition, ISBN 9780071485470, 2008, 1328 pages.

[13] P. Janku, R. Dosek, R. Jasek, Obstacle Detection for Robotic System Using Combination of Ultrasonic Sensors and Infrared Sensors, Modern trends and Techniques in Computer Science, ISBN 978-3-319-06740-7, 2014, pp. 321-330.

[14] A. Richards, J. Scheer, W. Holm, Principles of Modern Radar, 2014, ISBN 978-1-61353-032-0, 891 pages.

[15] L. Shi, H. Wang, The complexity of problems in wireless communication, Telecommunication Systems, published online in October 2016 at http://link.springer.com/article/10.1007/s11235-016-0243-6, pp. 1-9.

[16] L. Stastny, L. Franek, P. Fiedler, Wireless communications in smart metering, 12th IFAC Conference on Programmable Devices and Embedded Systems, The International Federation of Automatic Control, September 25-27, 2013. Velke Karlovice, Czech Republic

[17] R. Möbus, U. Kolbe, Multi-target multi-object tracking, sensor fusion of radar and infrared, 2004 IEEE Intelligent vehicles symposium

[18] C. Connolly, Collision Avoidance Technology: from parking sensors to unmanned aircraft, Sensor Review, Vol. 27, 2007, pp. 182-188.

[19] P. Ripka, A. Tipek, Modern Sensors Handbook, 2007, ISBN 978-1-905209-66-8, 536 pages.

[20] S. Nof, Handbook of Automation, 2009, ISBN 978-3-540-78830-0, 1812 pages.

[21] A. Eskandarian, Handbook of Intelligent Vehicles, 2012, ISBN 978-0-85729-084-7, 1599 pages.

[22] M.Hrubos, J.Svetlik, Y.Nikitin, R.Pirnik,D.Nemec,V.Simak,A.Janota, J.Hrbcek, M.Gregor, Searching for collisions between mobile robot and environment, International Journal of Advanced Robotic Systems, September-October 2016, pp. 1-11.

[23] D. Sun, A. Kleiner, B. Nebewl, Behavior-based multi-robot collision avoidance, IEEE International Conference on Robotics&Automation, Hong Kong, China, 2014.

[24] L. Zeng, G. Bone, Mobile Robot Collision Avoidance in Human Environments, International Journal of Advanced Robotic Systems vol. 10, 2013, 15 pages.

[25] S. Ali Masoud, A. Masoud, Constrained motion control using vector potential fields, IEEE transactions on System Man and Cybernetics part A: Systems and Humans, vol. 30, June 2000, pp. 251-272.

[26] T. Kröger, M. Wahl, Advances in Robotic Research, ISBN 978-3-642-01213-6, 2009, 358 pages.

[27] M. Shah, Fundamentals of Computer Vision, eBook available at crcv.ucf.edu/gauss/BOOK.PDF, 1997, 132 pages.

[28] N. Verma, A. Siddhant, P. nama, A. Raj, A- .Mustafa, N. Dhar, A. Salour, Vision Based Obstacle Avoidance And Recognition System, 2015 IEEE Workshop on Computational Intelligence: Theories, Applications and Future Directions, ISBN 978-1-4673-8215-1, 7 pages.

[29] V. Subramanian, T. Burks, A. Arroyo, Development of machine vision and laser radar based autonomous vehicle guidance systems for citrus grove navigation, Computers and Electronics in Agriculture vol. 53, 2006, pp. 130-143.

[30] X. Wang, J. Tang, J. Niu, X. Zhao, Vision-based two-step brake detection method for vehicle collision avoidance, Nuerocomputing, Vol. 173, part 2, 15.01.2016, pp. 450-461.

[31] A. Souliman, A. Joukhadar, H. Alturbeh, J. Whidborne, Real Time Control of Multi-Agent Mobile Robots with Intelligent Collision Avoidance System, Science and Information Conference 2013 7-9.10.2013, London, United Kingdom, pp. 93-98.

[32] I. Iyidir, F. Tek, D. Kircah, Adaptive Visual Obstacle Detection for Mobile Robots using Monocular Camera and Ultrasonic Sensor, ECCV 2012 Workshops and Demonstrations, Part II, LNCS 7584, 2012, pp. 526–535.

[33] R. Katz, Towards the Development of a Laserscanner-based Collision avoidance system for trams, 2013 IEEE Intelligent Vehicles Symposium, 23-26.6.2013, Gold Coast, Australia, pp. 725-729.

[34] T. Dhanabalu, S. Sugumar, S. Suryaprakash, A. Vijay-Anand, Sensor based identification system for Train Collision Avoidance, IEEE Sponsored 2nd International Conference on Innovations in Information Embedded and Communication Systems ICIIECS'15, ISBN 978-1-4799-6818-3, 2015, 4 pages.

[35] C-P. Tsai, C-T. Chuang, M-C. Lu. W-Y. Wang, S-F. Shu, S-L. Chang, Machine-vision based obstacle avoidance system for robot system, ICSSE 2013, IEEE International Conference on System Science and Engineering, July 4-6, 2013, Budapest, Hungary, pp. 273-277.

[36] J. Wang, Q. Chen, Obstacle Detection Model Implementation Based On information Fusion Of Ultrasonic and Vision, Proceedings of The 2016 IEEE International Conference on Real-time Computing and Robotics, June 6-9, 2016, Angkor Wat, Cambodia, pp. 216-220.

[37] R.Aufrere, J. Gowdy, C.Mertz, C. Thorpe, C-C. Wang, T. Yata, Perception for collision avoidance and autonomous driving, Mechatronics 13 (2003), pp. 1149–1161.

[38] A. Cherubini, F. Chaumette, Visual Navigation of a mobile robot with laser-based collision avoidance, The International Journal of Robotics Research, 32(2), 2013, pp. 189-205.

[39] P. Venuvinod, W. Ma, Rapid prototyping: Laser-based and Other Techniques, ISBN 978-1-4757-6361-4, 2004, 344 pages.

[40] Pepperl-Fuchs R2100 datasheet, http://www.pepperl-fuchs.us/usa/en/classid_53.htm?view=productdetails&prodid=62235#overview (obtained 31.3.2017)

[41] R. Zurawski, The Industrial Communication Technology Handbook, ISBN 9781482207323, 2005, 1756 pages.

[42] Leddartech M16 datasheet, http://leddartech.com/download/datasheet-leddar-m16-multi-element-sensor-module (obtained 31.3.2017)

[43] Intel Joule datasheet, http://www.intel.com/buy/us/en/product/emergingtechnologies/intel-joule-570x-developer-kit-541737#tech_specs (obtained 31.3.2017)

[44] Nexcom NISE 91 Datasheet, https://www.microtron.be/wp-content/uploads/2016/11/Nexcom-NISE-91.pdf (obtained 25.4.2017)

[45] Inico S1000 User Manual, http://www.inico-tech.com/doc/S1000%20User%20Manual.pdf (obtained 18.4.2017)

[46] Arduino Uno datasheet, https://www.arduino.cc/en/Main/ArduinoBoardUno#techspecs (obtained 18.4.2017)

[47] National Instruments System-on-a-module datasheet, http://www.ni.com/pdf/embedded/NI_System_on_Module_Flyer.pdf (obtained 18.4.2017)

[48] Raspberry Pi 2 Model B product page, https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ (obtained 18.4.2017)

[49] A. Tellaeche, I. Maurtua, A. Ibarguren, Use of machine vision in collaborative robotics; An Industrial Case. ISBN 978-1-5090-1314-2 Emerging Technologiesand Factory Automation (ETFA), 2016 IEEE 21st International Conference on 6-9. Sept. 2016, 6 pages.

[50] Nodepccc package at npm, https://www.npmjs.com/package/nodepccc (obtained in May 2017).

APPENDIX A: SEQUENCE DIAGRAM OF SAVING THE ROUTE CHART

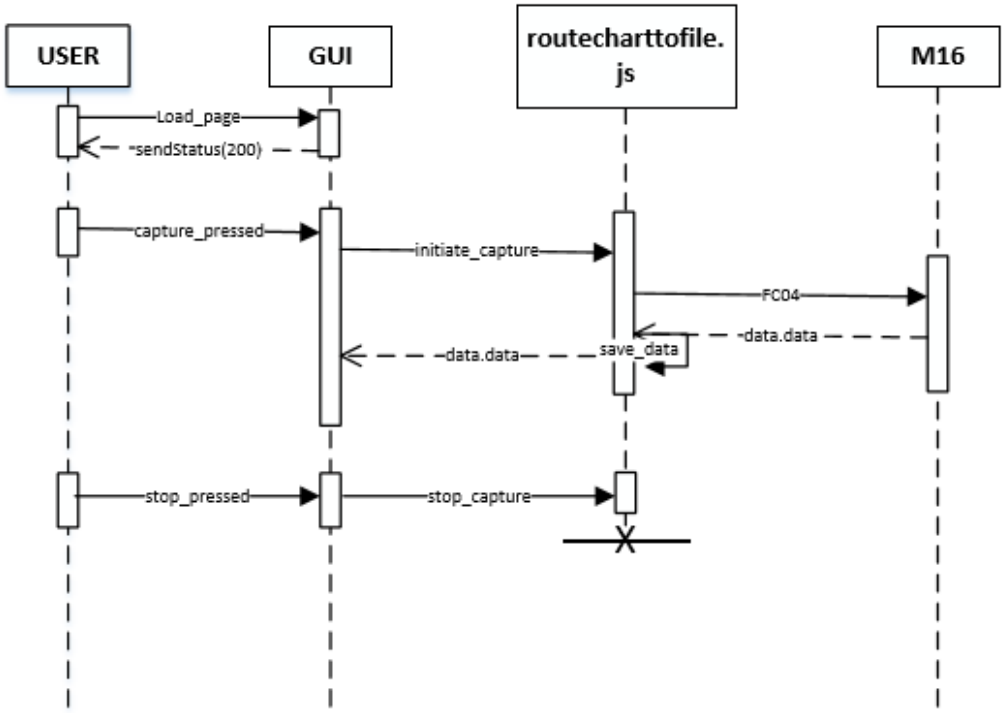


Figure 47: *UML diagram of route chart capture.*

APPENDIX B: SEQUENCE DIAGRAM OF ROUTE CHART COMPARE

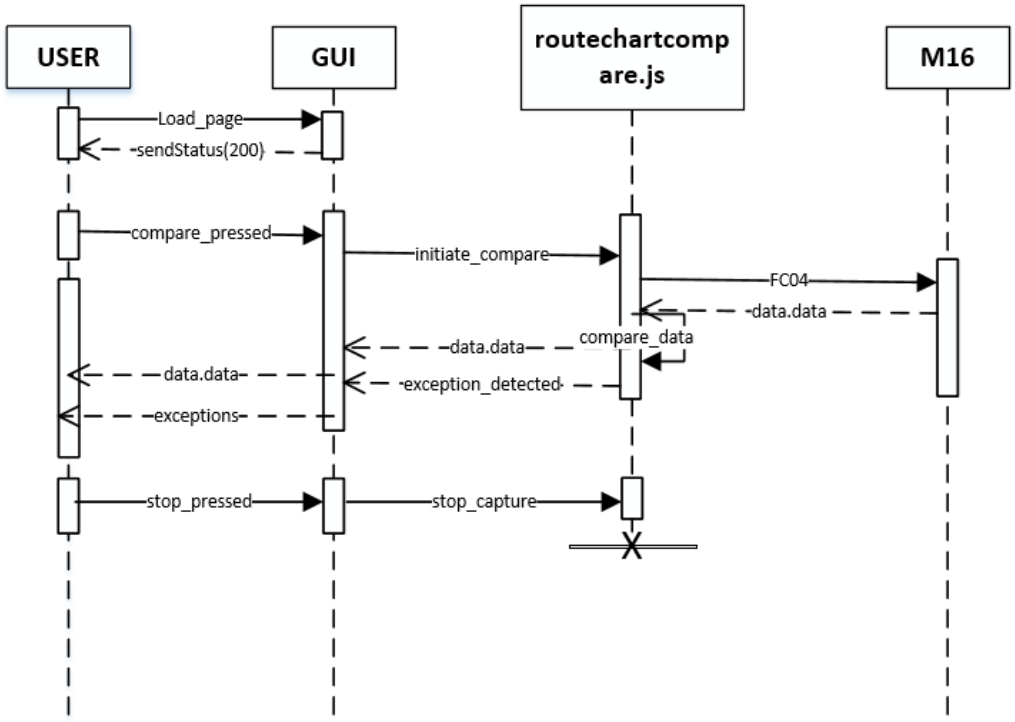


Figure 48: *UML diagram of comparing the live data to the route chart.*

APPENDIX C: USE CASE DIAGRAMS

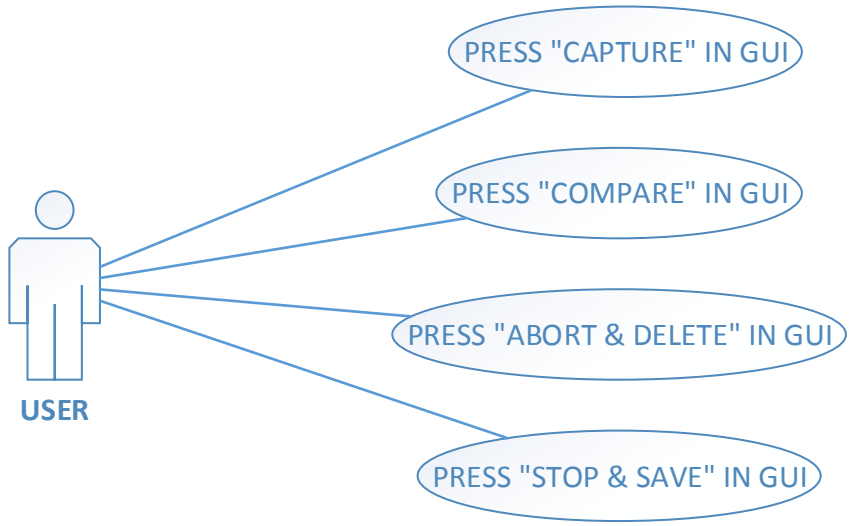


Figure 49: *Use case diagram 1: choosing the operation.*

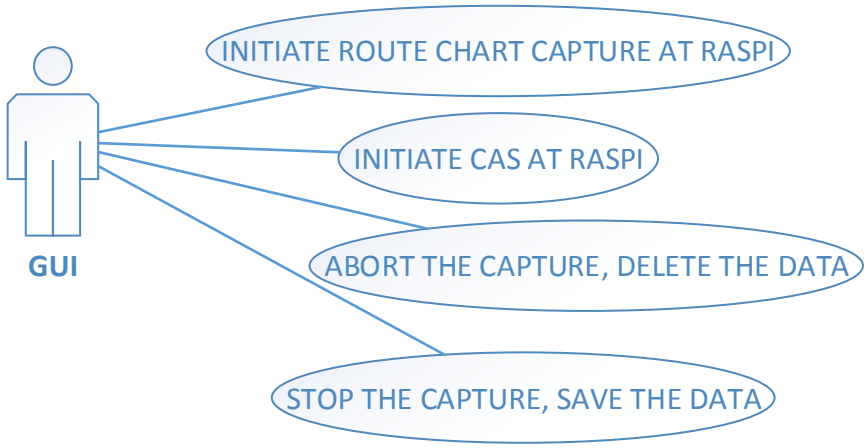


Figure 50: *GUI forwarding the user's choices to the CAS.*

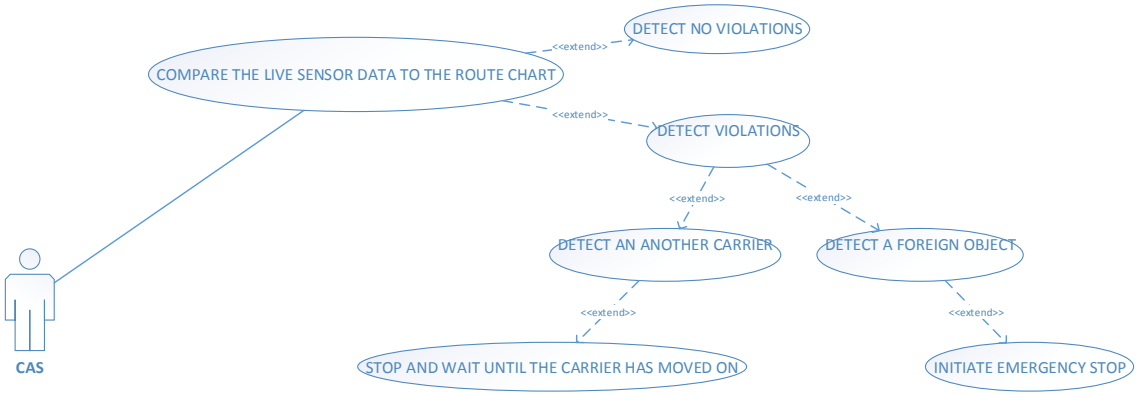


Figure 51: *Collision Avoidance System, simplified.*