High speed detecting and identification for car charging on electric roads

IULIANA STOICA AND VIKTOR NYBOM

MASTER´S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY |
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

High speed detecting and identification for car charging on electric roads

Iuliana Stoica And Viktor Nybom

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering (IEA), Lund University

Department of Electrical and Information Technology (EIT)
Lund University

Supervisors:
Lars Lindgren (IEA)
Fredrik Tufvesson (EIT)

Examiners:
Mats Alaküla (IEA)
Mats Gustafsson (EIT)

March 2, 2017

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

The constantly increasing awareness of protecting the environment has put electrical roads in the spotlight as an alternative solution to fossil driven means of transport. Dan Zethraeus has developed an innovative idea for a prototype electrical road which conductively supplies power to the cars whilst driving. The concept is to place a line of short rail segments in the middle of the drive lanes where each rail can have either grounded or positive polarity. The aim of this thesis work is to find solutions for the timing, detection and identification of cars so that the positive conductive rails are switched on correctly. The possible electromagnetic interference from the road is to be investigated and the communication methods adjusted accordingly. Finally, a demonstrator is built as a proof of concept for illustrating and testing the presented solution.

This report starts by presenting possible theoretical solutions for the detection and identification. Experiments that are set up to further analyse the most promising methods, and also the construction of the electronics for the detection and identification modules of the demonstrator follow. Furthermore, a simulation setup for analysis of the electromagnetic interference is tested. The complete solution and the whole setup of the demonstrator is presented in the last part. Results are presented for the performance of the demonstrator when tested on a real car driving at 30 km/h.

Terminology

EMI - electromagnetic interference
RFID - radio frequency identification
RSU - radio station unit
TSS - Traffic Supervisions Systems

Contents

Chapter 1

Introduction

1.1 Background

There is a consensus that greenhouse gas emissions must be reduced in order to keep the planet in the stable state which we are used to Oreskes [2004]. One way to reduce emissions is by working towards using renewable energy. About a quarter of the energy usage in Sweden comes from the transport sector and road traffic stands for 94% of this energy consumption [Ene, 2013, p.32]. Most of this traffic is still in need of fossil fuel and the swedish government has the goal of a 'fossil-free vehicle fleet' as of year 2030. [fos, 2013, p.35]. The inventor Dan Zethraeus is working on an idea for a prototype electric road, called ElOnRoad, with the aim of lowering the need of batteries and the need to stop and recharge electric vehicles. If the whole vehicle fleet can be driven by electricity then the problem has been minimized to that of making fossil-free electricity in the power-grid. Dan Zethraeus' idea is to place a line of short rail segments in the middle of the traffic lanes and have conductive sliding contacts mounted underneath the vehicles. The rail segments are placed so that every second segment is connected to ground as a negative polarity terminal and every other segment can be switched between the grounded negative polarity and a positive polarity. In this way a car with three sliding contacts and rectifiers can get constant DC power as shown in figure 1.1. The rail segments are thought to be one meter in length and have a short isolated area between each other. This master thesis aims to solve the timing, detection and identification of cars so that the switching of the positive conductive rails is done correctly. A demand on the system is that there should only be rails turned on underneath a correctly equipped electric car.

1.2 Related work

There have been several master theses involved in the ElOnRoad project prior to this one.

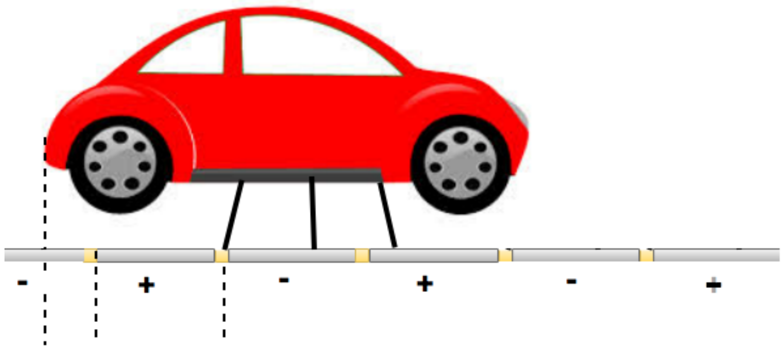


Figure 1.1: Electric road giving constant DC power to a car with three sliding contacts passing over it.

Philip Abrahamsson Completed his master thesis [Abrahamsson, 2015] on a first overall design. Part of the focus was on simulating the magnetic properties and design of over-voltage protection using LT-spice, Matlab and FEMM. Abrahamsson also did simulations on lightning strikes hitting the road and what effects this would give. Abrahamsson's results on magnetic properties have been background knowledge in our work, although the geometry doesn't fully align.

Marcus Andersson Completed his master thesis [Andersson, 2014] with focus on the switching circuit in the rails. Andersson was also part of building and fitting all the electronics in a full-scale proof of concept.

Henrik Fritzon Sund Completed his master thesis [Sund, 2014] with focus on detecting a vehicle and creating a control system for activating the switching of an individual rail. Sund's work has been of great help to us and a great part of our master thesis is a further development of his work and thoughts.

Filip Lillevars had not fully completed his master thesis before we wrote ours but given support in first hand on our work.

Emil Landqvist & Theodor Hallerby Completed their master thesis [Landqvist and Hallerby, 2015] with focus on developing a more comprehensive model of the road. The main focus has been on the thermal aspects and analysing overheating in different environments.

There are a couple of other electric road projects emerging in Sweden. The two biggest are shortly mentioned here.

Elways [elw, 2015] Is a system that allows both heavy and light electric vehicles to charge while driving. The system consist of a rail in the road, with sliding slots. An arm on the vehicle can grab on to the rail and have a conductive transmission of power.

eHighway [sie, 2016] Is a system that lets trucks and high vehicles to charge while driving. In this system there are power lines hanging over the road and the trucks have pickups mounted on the roof. Quite similar to a the electric train network.

Other work found to be related to this master thesis is the use of RFID in train systems and car positioning described in section 3.2.1.

1.3 Aim of this thesis work

The aim of this thesis work is:

1. To come up with a theoretic solution to uniquely identify a correctly equipped car, traveling at speeds between 20 km/h and 200 km/h and to determine when to activate and deactivate the positive one meter rail segments in the electrical road ElOnRoad. This should have enough accuracy so that a rail is only active when it absolutely needs to be and no high potential parts point out from under the passing car.
2. To identify and determine the possible electromagnetic interference from the road and compensate the communication for this.
3. To build a demonstrator as proof of concept capable of identifying, positioning and correctly activating something at a speed of around 50 km/h.

1.4 Limitations

Some parts of the combined and full solution as well as some minor parts of our work have been left out since it could hinder Dan Zethraeus from protecting sensitive parts of his invention.

Identification methods using cameras or lasers are not considered in this report since the environment around a road can get wet and dirty and components mounted on a car are subject to high tear.

This master thesis aims at designing a proof of concept turning on LED lights instead of a power giving rail. This thesis does not aim at finding or using components or optimizing code for use in a real prototype with demands on heat and durability.

It is in this report expected to be sufficient length between the ends of the car and the sliding pickup contacts relative to the length of a rail to make sure it is possible to have no active parts peek out from under the car.

In between standstill and 10 km/h are considered special conditions and will not be the focus of this thesis work.

Chapter 2

Approach

2.1 Approach and dividing the problem

The aim of this master thesis is to both identify a specific car and to accurately determine a defined position of the car in order to switch the rail on and off with the correct timing. As a first approach, the idea of using RFID (radio frequency identification) and the inductive detection circuit built by Fritzon were further examined, [Sund, 2014, p.5-17]. After gaining some understanding of how RFID systems work (see section 3.1.1), it was decided to handle the theoretical solution for the identification and precise positioning separately. Different possible ways to determine the position of the car are presented in section 3.2. After this an analysis of possible ways to combine the solutions with RFID are discussed in section 3.3. The chosen solution is then tested and analyzed in chapter 4 where finally a complete test of the proof of concept is carried out and described in section 4.4.

2.2 Finding information

A lot of knowledge and know-how from courses attended at LTH have been of use to us working with this thesis project. Most of the knowledge and information about radio transmission and RFID are gathered from the RFID Handbook [Finkenzeller, 2010] and from various research articles found on the Internet. Our supervisor Lars Lindgren has been a great help to most of our other needs of finding and gaining information and knowledge about everything from EMI (electromagnetic interference) to design and measurements.

Chapter 3

Theoretical solutions

The overall idea for the system, shown in Figure 3.1, is to use a two-way communication link between each car and a Radio Station Unit (RSU) marked as *A*. Between each charging rail there is a possibility to place a radio detector or receiver, marked as *C* in the picture. This since the material in the isolation part is not blocking radio waves and magnetic fields as the material in the conductive rails does.

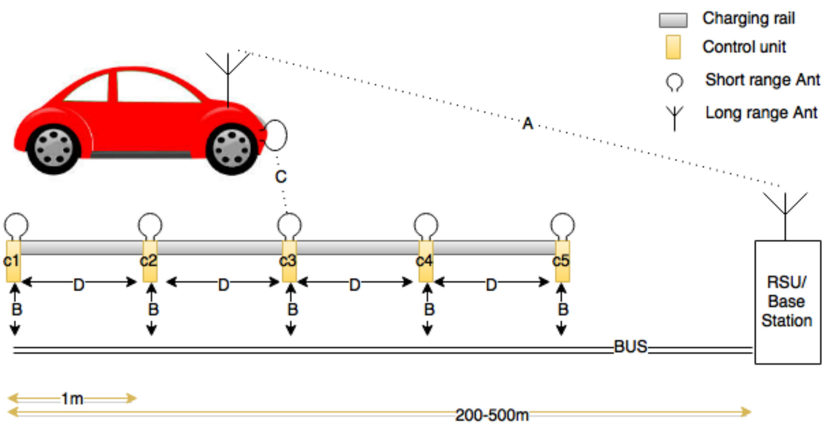


Figure 3.1: Sketch of the overall idea of the system.

3.1 Solutions for car identification

Methods presented in this chapter are based on RFID (radio frequency identification) technology which is more suitable for this application.

3.1.1 RFID technology

A RFID system consists of two main parts, the interrogator, or the reader, and the transponder which is located on the object to be identified [Finkenzeller, 2010, p. 6]. The reader generates a radio signal and when a transponder is located within the reader's range and can pick up the signal it gets activated and starts data exchange. The transponder consists of a coupling element and a microchip where data is stored and modulation is done. A transponder can be passive if it gets power supplied by the field generated by the reader or active if it has its own power source.

Communicating data can be done in full-, half-duplex or sequential mode. In full-duplex mode both sides are sending and receiving data simultaneously, whereas in half-duplex and sequential mode reader and transponder are taking turns on sending data. Transfer of energy from reader to transponder is continuous in duplex mode but it occurs only in between the messages in sequential mode [Finkenzeller, 2010, p. 39]. All digital modulation techniques can be used to transmit data but amplitude shift keying is the one most commonly used. There are a few coupling methods,i.e ways of energy transfer between reader and transponder. Based on reasonable distance ranges for this applications and availability of products on the market this report will only review inductive and backscatter coupling [Finkenzeller, 2010, p. 22, 45].

A quick review of radio waves and the different kind of fields that can be found around an antenna is given for a better understanding of the coupling methods. The regions surrounding an antenna are usually divided into three zones: reactive near-field, radiating near-field (Fresnel) and far-field (Fraunhofer) [Balanis, 2005, p. 34]. The transition boundaries between these regions are gradual but there are general approximations that work for most antennas. The transition from reactive near field to radiating near field and far field is usually approximated by the transition distances r_1 and r_2 [Balanis, 2005, p. 34]:

$$r_1 = 0.62 \cdot \sqrt{D^3/\lambda}$$
$$r_2 = 2D^2/\lambda,$$

where D is the largest dimension of the antenna and λ is the wavelength, see figure 3.2. Note that D must be much larger than the wavelength for these boundaries to be valid. For very short dipole antennas, where the largest dimension of the antenna is less than the wavelength, the transition out of the reactive near-field is according to Balanis often approximated by:

$$r_1 = \lambda/2\pi, \tag{3.1}$$

Radio waves are electromagnetic radiation created by accelerating charges. This movement of charges generates electric and magnetic fields which together form electromagnetic fields. In the reactive near-field, which is the region closest to

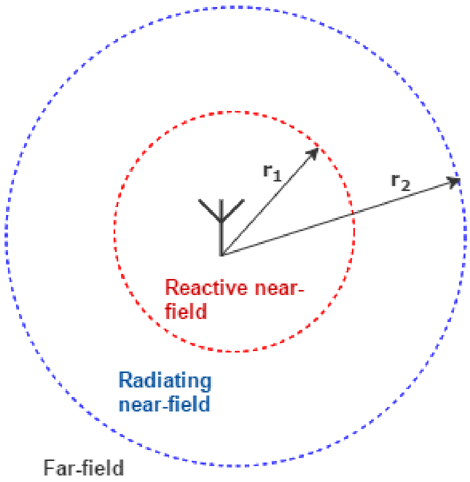


Figure 3.2: Field regions of an antenna, where r_1 and r_2 indicate the transitions into the reactive near-field and the far-field respectively.

the antenna, the electric and magnetic fields have greater magnitudes and varying phases. Also the pattern of the electromagnetic wave is not fully formed yet and the electric and magnetic fields decay with the square and cube of the distance to the antenna respectively [Rudge et al., 1982, p.13]. In the radiating near-field region, the radiating fields dominate but the pattern still varies with the distance. And finally in the far-field the electric and magnetic fields are orthogonal to each other and the radiation pattern of the electromagnetic waves decays linearly with the distance which means it does not significantly vary with distance anymore [Rudge et al., 1982, p.13].

3.1.1.1 Inductive coupling

Inductively coupled RFID systems usually have a passive transponder which is energized by magnetic fields in the near-field of the antenna [Evdokimov et al., 2010, p.7]. On the transponder side there is an LC circuit that resonates at a specified frequency, the coil is the coupling element and works as an antenna. The reader generates an alternating magnetic field which induces a voltage in the transponder's antenna. This voltage is rectified and will be the power supply to the microchip, see figure 3.3.

The induced current will have opposite sign and act against the generating magnetic field according to Faraday's induction law. This means the induced current creates a magnetic field of its own which induces a voltage in the generator coil on the reader side causing a voltage drop and thus a weakening of the magnetic field strength on the reader side. By switching a load resistor on and off the transponder can change the magnitude of this voltage drop and let the switch

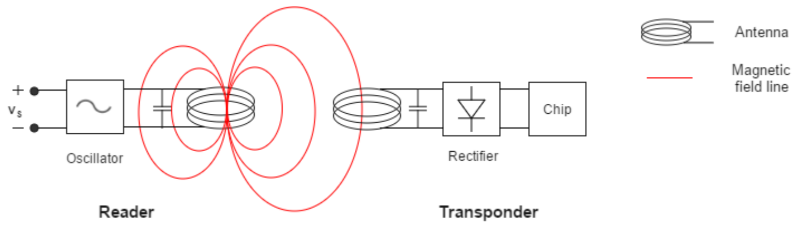


Figure 3.3: Sketch showing the principle of inductive coupling

timing be controlled by the data to be sent. Transmitting data this way is called load modulation, [Finkenzeller, 2010, p.40-43]. Inductively coupled systems can operate in the LF (low frequency) range usually around 125 kHz with a range of up to 0.5 m and HF (high frequency), usually 13.56 MHz with a range of up to 1 m, [Evdokimov et al., 2010, p.7]. The near field at 13.56 MHz is roughly limited to 3.5 m according to 3.1 mentioned earlier. Practical limitations do however reduce the range to about 1 m. Range is thus one of the main factors that affect the frequency choice and antenna dimensions for an RFID system. One disadvantage with inductive coupling is susceptibility to electromagnetic disturbances such as those generated by welding robots or strong electric motors [Finkenzeller, 2010, p.26].

3.1.1.2 Backscatter coupling

In backscatter coupling the reader's antenna sends electromagnetic waves that are reflected back by the transponder antenna. The way in which the electromagnetic waves are reflected back depends on the properties of the transponder such as cross sectional area and antenna characteristics. The reflection cross-section can be altered by a load connected to the transponder's antenna and thus data can be sent by modulating the reflected amplitude.

Since backscatter systems operate in the far-field of an antenna they usually have a range of more than 1 meter and the most common operating frequencies for backscatter coupling are 868 MHz (Europe) and 915 MHz (USA) or the microwaves band. [Finkenzeller, 2010, p.156]. Because of the long range the radiated power is low so the transponder usually is of active type. Data transmission still relies exclusively on the power in the electromagnetic field emitted by the reader but is according to Finkenzeller more robust against electromagnetic disturbances than inductive coupled systems, [Finkenzeller, 2010, p.45-48].

3.1.1.3 Applications

RFID technology has been widely used in the transport sector for speed measurements, vehicle counting, wagon tracking, updating time tables for bus and train arrivals, electronic toll collection (ETC) and more [Xiaoqiang and Manos, 2011].

This section will present a few examples of relevant high-speed RFID applications and experiments.

LF spectrum

Most of the RFID applications in the LF (low frequency) spectrum have an operating frequency of 125 kHz. TSS (Traffic Supervisions Systems) is a company which has developed solutions for several ITS (Intelligent Transportation System) applications for roads and railways. Their systems can be structured into two categories: AVI (Automatic Vehicle Identification) and AVL (Automatic Vehicle Location). In AVI systems the tags are placed on the vehicle and readers in the infrastructure. In AVL systems the reader is mounted on the vehicle and tags are placed at specific positions in roads/railways, each tag corresponding to a location registered in a database. The read tag ID can for example be sent to a central information system via radio or just used by the car processing unit for different purposes.

One of the company's AVL systems, called TPL (train position locator), has interesting specifications. It has a speed range of 0-300 km/h with a position accuracy of ± 1 cm at low speeds and ± 50 cm at the maximum speed, [TSS, 2015]. The antenna dimensions are 108x335x59 mm and the tag housing has a diameter of 40 mm and a length of 330 mm. The EMI environment under a train should have similarities to the one under a car getting power supplied conductively driving on an electrical road. The specifications of this system makes it very interesting for the project.

HF spectrum

RFID systems with an operating frequency of 13.56 MHz are said to be working in the HF (high frequency) spectrum. Optys Corporation, an RFID design and developing company has done high-speed reading tests with a 19.5 cm long tag and 30x10 cm antenna at an operating frequency of 13.56 MHz. The tag was attached on top of a H0 scale model train and the antenna was suspended above the rails. When moving at the speed of 113 km/h the tag was successfully read once. Unfortunately a documentation report could not be found but a video of the experiment can be watched at [Optys Corporation, 2011]. The distance between the reader and the tag is not specified but judging from the video it may be roughly 10-15 cm. The standard used in this system is ISO 15693 which includes basic communication protocols and anticollision algorithms which allow tags to take turns in communicating with the reader.

UHF spectrum

The standard called EPC Global Gen2 specifies RFID systems with a range of 2.5 cm - 10 m and operating frequencies in the UHF (ultra high frequency) spectrum

(858 - 930 MHz). Each RFID tag, active or passive, contains an universal identifier in the form of an EPC code (electronic product code) saved on the memory chip.

This protocol is built to suit inventories in warehouses where many products have to be identified fast and information such as shelf life, shipping date have to be read/written to a tag. Therefore the reader has to go through these three states: select (where a population of tags are selected for inventory), inventory (where each tag is identified by its EPC) and access (where the reader can read/write to the tag's memory chip). The tag goes through the following states for communication: ready (waiting to be selected for an inventory), arbitrate (has been selected but waits for its turn to identify itself), reply(identifies itself), acknowledged (identification succeeded), open and secured (access to the tag's memory), [epc, 2008, p.45-48].

Harting Technology Group (HTG) has done high-speed tests using their SL89 RFID tag, RF-R500-p-EU reader and WR80-30 antenna. The antenna's operating frequency is specified to 902-928 MHz and the tag uses the standard EPC Gen2. The tag was mounted on the car facing the side of the road where the reader was placed at a distance of 2.5 m at the passing point. When driving past the antenna at 200 km/h HTG got 9 correct readings of the 96 bit EPC identifier. Using their smaller Ha-VIS RF-ANT-WR30-EU antenna and Ha-VIS RF-R500-c-EU reader HTG still got 1 correct reading at 200km/h. Further specifications of the experiment and a video can be found at [Wermke et al., 2014a], [Wermke et al., 2014b].

Another highly interesting application is ATIS (Automatic Train Identification System) which is primarily used in identifying cargo trains usually traveling at speeds less than 100 km/h, [Xiaoqiang and Manos, 2011]. This system has been in use in China since 2007. A UHF reader is placed between the rails and a passive tag is mounted on the train and its reading distance is specified to about 1.4 m. Since it would be unnecessary for the reader to constantly be turned on, magnetic steel detection systems are placed at distances of 40-50 m from the reader to send turn-on and turn-off signals to the reader when trains are approaching or leaving the area. When a train is approaching, the reader gets turned on and the read tag ID is sent further to a central information system which can update time tables for example. The reader is then turned off when the train has activated the other magnetic steel detection system. The system also includes a module which can write information to tags.

It seems like there is a large focus on research regarding the adjusting of such a system to modern high-speed and ultra high-speed trains which can reach a velocity of up to 500 km/h. The biggest challenges are the tag latency and the fact that the tag is within the reading range for a very short period of time for very high speeds, [Xiaoqiang and Manos, 2011].

Choosing frequency and protocol

The reading distance needed between reader and transponder will decide the appropriate coupling method to be used. Power losses in the environment in which the system is meant to be used, the regulations for the allowable radiated power specified for each frequency range and limitations on antenna sizes give further constraints.

As mentioned before the power of the magnetic field decays with the cube of the distance in the near-field which corresponds to 60 dB/decade. Thereafter the power of the electromagnetic field decays linearly with distance in the far-field which corresponds to 20 dB/decade. The specifications for the maximum allowed transmit power is usually given as the field strength at a distance of 10 m from the reader. Lower frequencies have a larger near-field region according to equation 3.1 which means a higher initial transmit power will decay to the same strength at 10 m as a lower initial power transmitted at a higher frequency, [Finkenzeller, 2010, p.162] This is an interesting optimization aspect when choosing to work with inductively coupled systems. A lower frequency means a slower reading speed though since there are fewer wave periods per unit of time to process. The reading range could instead be increased by using larger tags, and thereby get a longer time window and more time for the reading to take place.

The environment of an RFID system can cause problems. Metallic surroundings can affect the strength of the field between reader and transponder and thus reduce the reading area. This is due to eddy currents induced in the metal which oppose the initial field according to Lenz's law. There are many solutions to this problem, one of them being ferrite shielding where a piece of ferrite with high magnetic permeability is placed between the antenna and the metal. This will prevent eddy currents but the field strength may get higher so adjustments have to be made, [Finkenzeller, 2010, p.107-108].

An issue with using UHF RFID is that the operating frequency allowed varies and there exists one standard in the US and one standard in EU. To maximize the efficiency of the antennas of the reader these are often tuned to either the EU standard or the US standard and not both. So to use the UHF system in the whole world both antenna standards would need to be installed.

In an application such as identifying objects moving at high speeds a simple protocol would be the best suited. The extra functionality that EPC gen2 has is not needed in this case and would contribute to longer read latency, mostly because of the anti collision algorithms.

3.2 Solutions for car positioning

In this section the different methods that were considered for the the positioning are described. Since no electric active parts are allowed to be exposed in front or back of the car the positioning needs to be accurate to within centimeters, even at higher speeds. As the environment can get dirty and the electrical road is subject to wear and tear some methods as for example methods using photodetectors are not suitable for this system and will not be further discussed in this report.

3.2.1 Identifying antenna signal strength

This method requires two transmitter antennas on the car and one after each power delivering rail.

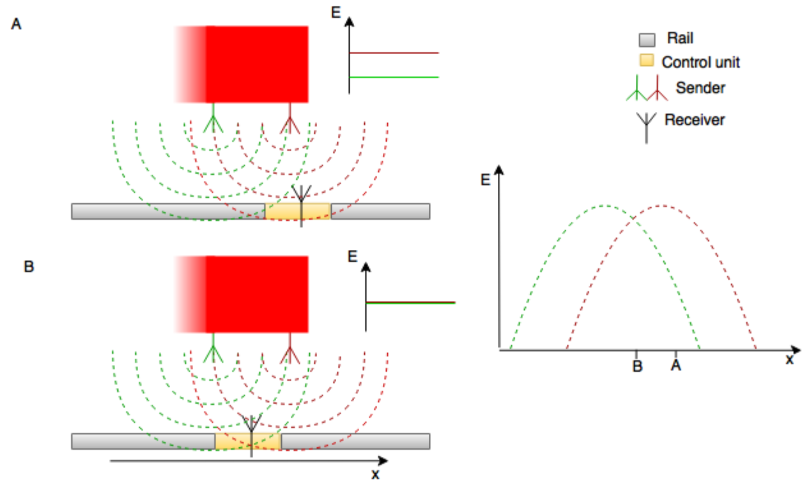


Figure 3.4: Car positioning using two antennas that send synchronized as seen in A. The third antenna (placed in the road) detects when the two signals have the same strength as seen in B.

The car's position is identified when the receiver antenna registers the same signal strength from both transmitter antennas, see figure 3.4. Trying to compare two signal strengths is problematic since the signals can get affected differently by the surroundings. Electromagnetic interference from the road can for instance affect the reception. Thus the amplitude of the received signals does not only depend on the distance from the car. This makes it difficult to determine when the two signals have the same amplitude. Since the positioning also has high time constraints there is a very short time window for processing the data more thoroughly in order to get higher accuracy. This idea is decided to be to problematic and not to be analyzed any further.

3.2.2 Doppler effect

The components required are one receiver antenna after each power delivering rail and one/two sending antennas on the car.

Having a radio transmitter on a moving car, the intercepted frequency will be higher than the source frequency when the car is approaching the receiver and lower when the car is moving further away due to the Doppler effect. Therefore the difference between the observed and emitted frequency will have a step form going from positive to negative if the transmitter is approaching the receiver directly. Using this method the position of the car is detected by identifying this frequency hop, see figure 3.5.

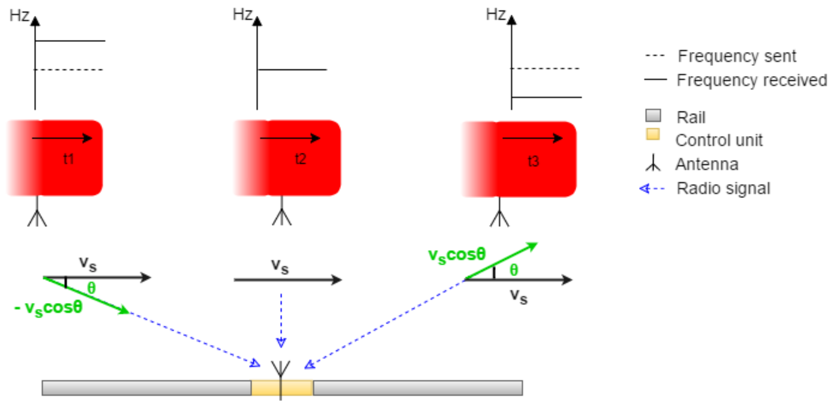


Figure 3.5: Car positioning by detecting the frequency hop caused by the Doppler effect. The car (marked as the red square) is moving from left to right with constant speed as indicated by the velocity vector v_s. At the time instances t1, t2 and t3 the received frequency will be higher, equal and lower than the sent frequency respectively.

The formula describing the relationship between the emitted frequency, f_s and the received frequency f_r for a stationary receiver is:

$$f_r = \frac{c}{\lambda} = \frac{c}{c \pm v_s} \cdot f_s,$$

where λ is the wavelength, c is the speed of light and v_s is the radial component of the speed of the transmitter, i.e the velocity component along a straight line between the transmitter and receiver, [Tipler and Gene, 2008, p.518-519]. Since the received frequency is higher when the transmitter is approaching, the wavelength will be shorter and that is when the velocity gets a negative sign as indicated by the formula.

In this case the receiver is approached at an angle because the transmitter is placed on the car at a height h from the road. The radial component of the velocity will thus not switch signs directly when passing the receiver but rather make a slower transition depending on the angle, see figure 3.5. This is the reason why the frequency hop does not change like a perfect step curve but it monotonically decreases instead. The effect of the height on the change in frequency can be seen in figure 3.6a for relevant heights between 10-30 cm, which is a reasonable range

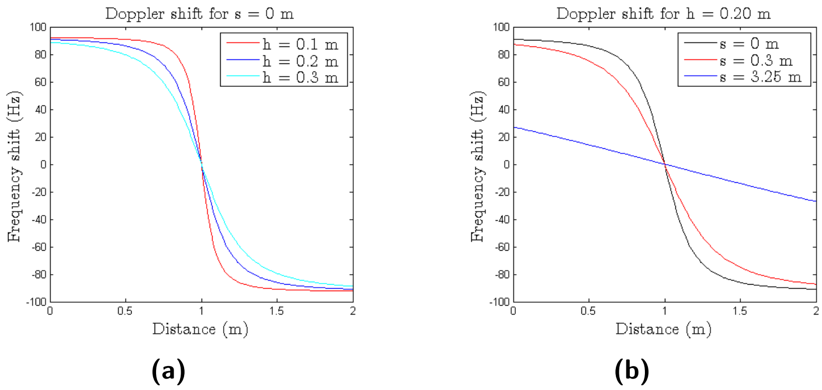


Figure 3.6: Doppler shift at 1 GHz and 100 km/h passing speed with detector placed after 1 m for a) different heights h of car ground clearance and b) different sideway positions s. The blue line corresponds here to the car being in the other lane.

for the ground clearance of a car. The figure also shows that the Doppler shift is about 200 Hz in total for a 1 GHz wave, if the car travels at 100 km/h. Moving laterally relative to the receiver will give less change in frequency, see figure 3.6b, where a movement of 3.25 m means that the car has switched lanes, considering that a typical bidirectional road has a width of 6.5 m, [VGU, 2004].

In the scenario with one transmitter antenna, this should be placed on the front part of the car in such a way that when the position has been detected the on-signal to next rail should be sent. The off-signal to the rail about to be left behind can then be calculated with the help of the speed and the distance between the antenna and the last pickup (assuming this is some standard or that it is communicated to the system by the car) and adding some time margin. In the scenario with two antennas, the additional antenna would be placed on the back of the car and would signal when a rail should be turned off. The latter scenario is more accurate in case the car is accelerating but also safer in case something goes wrong, like an accident, corrupt speed data or other errors in the system.

The problem with this method is once again the accuracy. The method first demands very stable frequency references for the signals of the transmitter and receiver antennas. Secondly as seen in picture 3.6a the accuracy of the frequency shift is very dependent on the height difference between transmitter and detector, making the method harder to realize for a fast and high accuracy detection.

3.2.3 Inductive detection

The principle of magnetic induction is a common way of detecting cars approaching drive-throughs and traffic lights. These detectors are usually called inductive-loop traffic detectors and consist of wire loops installed in the roadways and a processing unit. The loop is oscillating with a certain frequency which increases as a car passes over it because the large amount of metal on the car lowers the inductance of the loop. Changes in frequency are processed and give the detection signal.

Using magnetic induction in traffic applications has many advantages. It is a simple solution. It is very resistant to wear, tear and dirty environments. This method does not have enough accuracy to be used for identifying the position of a car on an electric road though. The idea of using magnetic induction and two resonant circuits has however been analysed and tested with good results in a previous master thesis [Sund, 2014] where a proof of concept was built to detect a car. The system, shown in figure 3.7, consists of a series LC circuit as a transmitter and a parallel LC circuit as a receiver. Both tuned at the same resonant frequency. The working principle is relying on the fact that alternating current flowing through the transmitter circuit generates an alternating magnetic field, which can induce a voltage in the receiver circuit. In a way, simplified, this is the same way that the RFID systems work. A parallel LC circuit is used on the receiver side because it has a high impedance at resonance while a series LC circuit has a low impedance which enables high current through it which is needed to create a strong magnetic field.

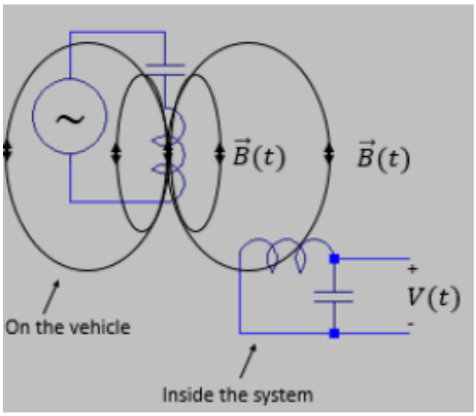


Figure 3.7: Illustration from earlier master thesis [Sund, 2014, fig 3, p.7] showing the differences of the resonance circuits at the transmitter and receiver.

At the resonant frequency an LC circuit is purely resistive which means the reactances of the inductor and the capacitor cancel each other out. The resonance frequency in an LC circuit is then given by:

$$2\pi f_0 L = \frac{1}{2\pi f_0 C} \Leftrightarrow f_0 = \frac{1}{2\pi\sqrt{LC}}, \tag{3.2}$$

where L and C are the inductance and the capacitance.

As a measure of efficiency in a resonant circuit, the quality factor Q is defined as the ratio between the energy stored in the circuit and the average power dissipated.

For a resonant circuit with an sinusoidal input signal with the angular resonant frequency ω, the expression is:

$$Q = \omega \frac{energy\, stored}{average\, power\, dissipated}.$$

The total energy in a resonant circuit is constant at resonance and oscillates back and forth between the inductor and capacitor. This means that the total energy stored in the system at any time is equal to the maximum energy stored in either the inductor or the capacitor. Since the system is purely resistive at resonance the average power loss is simply $P_{avg} = I_{pk}^2 R/2$, where I_{pk} is the peak current, [Thomas, 2004, p.87-92]. The Q factor in a series RLC circuit at resonance can then be derived as follows:

$$Q = \omega_0 \frac{E_{tot}}{P_{avg}} = \omega_0 \frac{\frac{1}{2} L I_{pk}^2}{\frac{1}{2} I_{pk}^2 R_s} = \omega_0 \frac{L}{R_s} = \frac{\sqrt{L/C}}{R_s}, \tag{3.3}$$

The formula for the Q factor in a parallel RLC circuit can be derived in a similar way to:

$$Q = \omega_0 R_p C = \frac{R_p}{\sqrt{L/C}}, \tag{3.4}$$

The Q factor is not only an indicator of efficiency but since it is a function of power losses it relates to the bandwidth and ringing in the circuit as well. Solving the equation $|H(\omega)| = 1/\sqrt{2}$ for ω gives two solutions, ω_1 and ω_2 which are the frequencies at which half power occurs. This gives the following bandwidth for the series and parallel circuits:

$$\Delta\omega = \omega_1 - \omega_2 = \begin{cases} R/L, & (series) \\ 1/RC, & (parallel) \end{cases}, \tag{3.5}$$

By putting eq. 3.3, 3.4 and 3.5 together it can be showed that the bandwidth in a resonant circuit is:

$$Q = \frac{\omega_0}{\Delta\omega}, \tag{3.6}$$

Factorising the transfer function and inverse transforming to get the impulse response in the time plane, gives an expression for the time constants for a parallel and series RLC circuit. The time constants are $2RC$ and $2L/R$. Using this together with eq. 3.3 and 3.4 gives the following relationship between the Q factor and the time constant in a resonant RLC circuit:

$$Q = \frac{\tau\omega_0}{2} \tag{3.7}$$

Thus the Q factor is a useful parameter when designing a resonant circuit not only as a measure of efficiency but also bandwidth and time constant. A higher Q value means a smaller bandwidth but a higher time constant which means the ringings in the oscillations will die out slower.

Sund's idea and ground work shows good potential and should be analyzed further. There is for instance a lack of experimentation and analyses of how accurate or fast the system is or could be. In his system a simple amplitude comparative method is used to detect the car [Sund, 2014, p.51-58] which leaves much to wish for in accuracy.

3.2.4 Conductive pickup signaling

Using the fact that the pickups have conductive contact with the rails and the power is transmitted as direct current it would be feasible to send a sinusoidal signal down from a pickup to a rail. If two pickups send two different sinusoidal

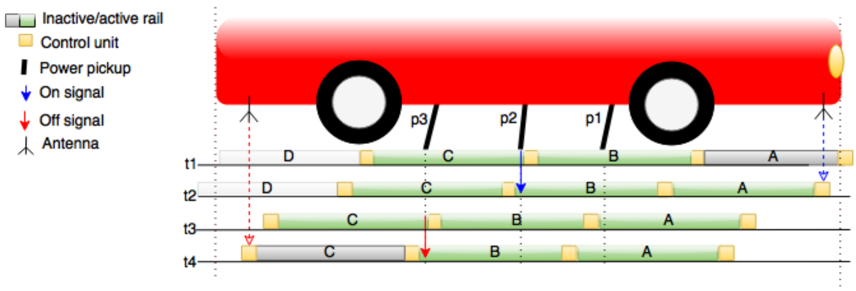


Figure 3.8: Positioning using pickup two (p2) and three (p3), each sending a sinusoidal signal with a specific frequency down to the rails. When p2 makes a jump from rail C to rail B the control unit turns on rail A. When (p3) makes a jump from rail C to rail B the control unit turns off rail C. In the figure there are also antennas to show the use of wireless positioning.

signals it would be possible to distinguish the pickups from each other as seen in figure 3.8. If one signal is detected at one of the rails by the control unit and then detected at the adjacent rail by the next control unit, it is said that a jump has occurred. The more accurately this jump is detected, the more accurately can the position be decided. At low speeds this jump would probably be detected directly as it occurs. As the pickups might bounce off the road it could get difficult to accurately detect the signal at high speeds. Challenging parts of this method are high pass filtering the signal from the power rails, and the need for the power rails and pickups to be high frequency isolated from neighboring pickups and rails.

3.2.5 Short circuit detectors on rail

Instead of sending and detecting a signal sent through the pickups it could be possible to design the rails so that a passing pickup could trigger short circuits placed at specific positions on the rail as seen in figure 3.9. In order to make the detection more robust against false positives, detection when no detection should occur, two short circuits at two different positions on the rail can be designed to occur at the same time. To make double detections occur when it is time to turn a rail on or off, four short circuit gaps have to be placed on the rail. As shown in figure 3.9, equidistantly placed pickups cause the short circuit detection pattern to include two double detections of each of the double detection gap pairs. This can be seen in figure 3.9 where for instance the times t1 and t4 have the same short circuit gap pairs. One way to make sure that the switching of a rail takes place

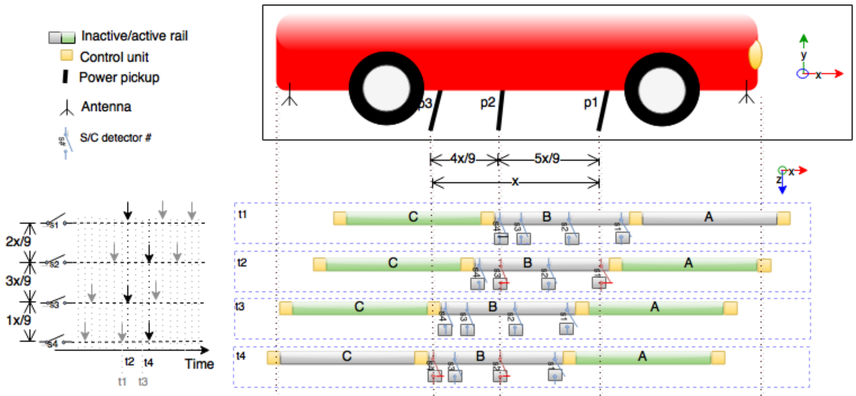


Figure 3.9: The case where the cars have equally spread pickups p1-p3. The rail is equipped with four gaps along z-axis with potential difference to the rail that the pickups short circuit when they move over them. The gaps are placed so that two pickups short circuit gaps at the same time when a rail should be turned on (t3) or turned off (t4).

at the correct detection is to have a state machine in the control unit code. This state machine goes through the states resembling the timelines in the graph shown in the lower left of figure 3.9 and therefore knows when the correct detection pair occurs.

Another way to distinguish between correct and false detection pairs is to move the middle pickup on the car a little to the side. This has been done in figure 3.10 where the middle pickup (p2) has been moved backwards and the positioning of the short circuit detectors on the rails has been moved. This gives rise to a new detection pattern as shown in the lower left of figure 3.10 that don't have doublets of the detection pairs as the previous method had.

These conductive short circuit methods might work using hall detectors instead and in that way eliminating the need for isolated conductive parts and the risk

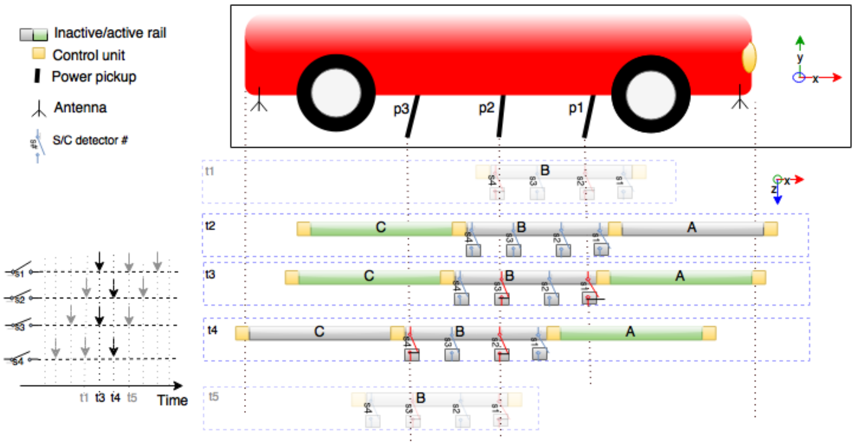


Figure 3.10: The case where the cars have equally spread pickups p1-p3. The rail is equipped with four gaps along z-axis with potential difference to the rail that the pickups short circuit when they move over them. The gaps are placed so that two pickups short circuit gaps at the same time when a rail should be turned on (t3) or turned off (t4).

of the pickup jumping over the detection area as well as easier rail design. This could be further investigated later.

3.2.6 Hall effect sensors

The concept is the same as in the short circuit method in 3.2.5, but with Hall effect sensors instead of short circuit sensors. By placing electromagnets on each side of pickup 3 and Hall effect sensors before each ground rail it should be possible to make an accurate detection.
The Hall effect is the phenomenon arising when an electric current passes through a metal located in a magnetic field. A potential proportional to the current and the magnetic field arises perpendicular oriented to them both. [Hall, 1879]. Hall effect sensors are commonly used for measuring rotational speed of motors. Often the sensors are of either a linear version or an on-off switch version. The analog sensors give an output proportional to the strength of the magnetic field whilst the switching version acts as a saturated transistor. For the sensors to work in the setup presented in 3.2.5 there can't be any ferromagnetic material placed between the magnet and the sensor which might be hard to achieve in the final design.

3.2.7 Sound and vibration

The frequency content in the sound wave that arises when the pickup has contact with the conductive material of the rail should be very distinct and different from that when the pickup has contact with the isolation material between the rails. It should be possible to determine when a pickup leaves and enters a rail using one or more vibration sensors in the area around the isolator parts. It might also be possible to analyze the sound and see distinctive changes in the vibrations depending on how many pickups that are on the same rail. If the hardware and the algorithm doing this can be made fast and cheap enough it is a possible way for precise detection. The method is however somewhat difficult to realize since no pickups have been designed or built yet to test this out on.

3.3 Different communication schemes

Different scenarios for the communication in the overall system have been briefly analyzed. Figure 3.11 is meant to help visualizing them all. The possible communication channels considered are a two-way communication link between each car and the Radio Station Unit (RSU) placed on the side of a road at appropriate intervals. The RSU uses some sort of radio communication, wifi or 3/4G, to communicate with each car, marked by A in the picture. Wired communication medium is used to communicate with the electronics in the road, marked by B in the picture. Between each charging rail there is a possibility for a radio detector or receiver, marked as C in the picture. There should also be a possibility for each control unit to communicate with its neighbor directly, marked as D in the figure.

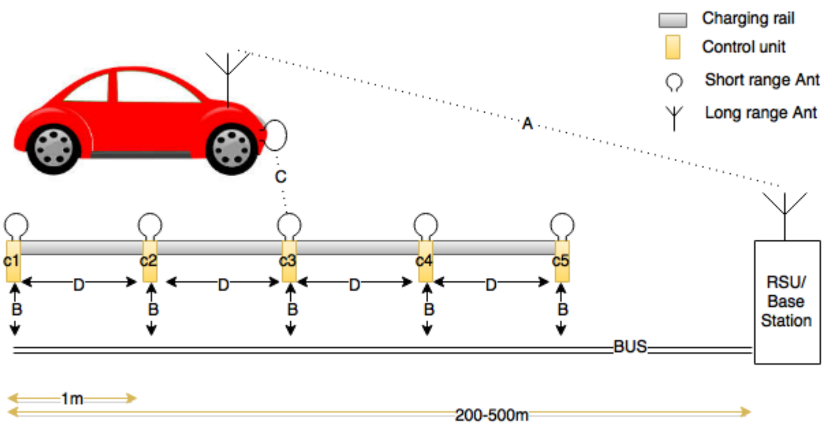


Figure 3.11: Illustrative sketch of the communication system to aid the understanding of the scenarios.

3.3.1 Scenario one

Using RFID tags between each road section and a RFID reader on the car, the car can read a tag on the road that corresponds to the next rail. The car can then communicate this tag ID together with its unique car ID to the RSU via channel

A. The RSU now has an uniquely identified car with a roughly accurate physical position. The RSU sends a signal to the corresponding rail control unit making it ready to switch the rail on and off. The switching occurs when a detection, using the possible ways described in *solutions for car positioning 3.2*, is triggered. In this scenario the intelligence is centralized to the RSU and the control units are only triggers controlling the switching of one rail. The RFID system is designed for use of many tags and few readers.

The scenario needs a fast working RFID system and a fast working long range communication so that the time from reading the RFID tag to switching the rail is not exceeded. If a car is traveling at a speed of 200 km/h and the RFID tag is read one meter before the switch is triggered, that gives a total time of 18 ms. Since the detection occurs at the rail control unit the communication between the car, RSU and rail control unit in this scenario doesn't have to be deterministic, just performed fast and reliably enough. Detection solutions thought of as appropriate for this scenario, out of the brief analyzing done in *solutions for car positioning 3.2* are Hall effect sensors and inductive detection.

3.3.2 Scenario two

This scenario is very similar to the previous scenario, except the RFID readers are placed in the road and a tag is placed on the car. In other words, this one way communication setup is the other way around from that in scenario one. The approaching car sends its own tag ID and a GPS position to the RSU. The RSU sends information about which tags are clear and accepted for use to all the rail control units in the vicinity of the GPS position. When the car tag ID is read by the rail control unit the position is roughly confirmed and the upcoming rail can be informed to get ready to be switched on, through communication link *D*. The accurate switching is again performed by one of the solutions from *solutions for car positioning 3.2*.

In this scenario the key corresponding to a car is the RFID tag which is constant and hard to keep secret. This makes it hard to identify which car is activating which rail in a satisfying manner. The RFID tags are also much cheaper than the readers, this scenario resulting in a more expensive road. On the other side this scenario doesn't have the same constraints on the communication speed for either channel *A* or *B*.

3.3.3 Scenario three

In this scenario the intelligence is distributed to the control units of each individual rail and there needs to be a two way communication link between car and control unit at each individual rail. The rail identifies itself with a RFID tag and the car sends a rail unique code that the rail uses to both identify and position the car. The long range channel *A* is used to negotiate the unique codes and other essential information. This scenario demands that the car is able to send a key and the speed, possibly around 32 + 8 bit, to a control unit at least 150 times per second for a radio range of around 0.5 m. This constraint is set by the maximum speed of 200 km/h and that a message should be sent at least three times to be received correctly.

There is no perfect way thought of in *solutions for car positioning 3.2* to send so much information from the car to the rail control unit and at the same time find a very accurate position at high speeds. Techniques that could be possible with more analyzing and investigation could be *Conductive pickup signaling*, *Identifying antenna signal strength*, *inductive detection* or *Doppler effect*. All of them might be hard to get to send enough information and at the same time accurately acquire the position.

Chapter 4

Analysis and testing

4.1 Car positioning

One of the most promising methods in *solutions for car positioning: 3.2* and also the one which could be tested immediately was the inductive method. An experimental setup to do comparing experiments was built and calibrated, then different antennas and frequencies were analyzed. Finally a detection system was built, implemented and tested outdoors.

4.1.1 Pendulum as experimental setup

In order to perform repeatable low speed tests in the laboratory a swing was built consisting of a rectangular wood plank suspended at the short ends as a swing. The length of the swinging pendulum measured from the pivot point to the center of mass of the plank was 258 cm and the plank itself had a length and width of 45 cm and 9.5 cm respectively. The height from the plank to the floor was about 10 cm as seen in figure 4.1a. The energy of a swing at any point is the sum of the potential and kinetic energy and this energy is conserved if the friction is neglected. Approximating the swing as a simple pendulum and setting up the energy balance equations gives the following expression for the pendulum's velocity at a height h:

$$E_{tot} = mgh_{start} = mgh + \frac{mv^2}{2} \Leftrightarrow v = \sqrt{2g(h_{start} - h)}$$

where m is the mass the of the pendulum, g is the gravitational acceleration, h_{start} is the height from which the pendulum is released and v is the velocity at the height h. According to this equation the velocity of the pendulum at the equilibrium point is 4.43 m/s (15.95 km/h) when released from a height of 1 m (from the floor). This was approximately the highest speed tested with since it was hard to manually release the pendulum from a higher height than this without it swinging sideways. For a more accurate velocity measurement and to get an accurate position reference a photodiode was placed underneath the swing and a strong light source was pointed from above. By measuring the time during which the swing shadowed the sensor the speed could be calculated, since the width of the wood plank shadow could be measured. The measurement circuit is shown in figure 4.1b.

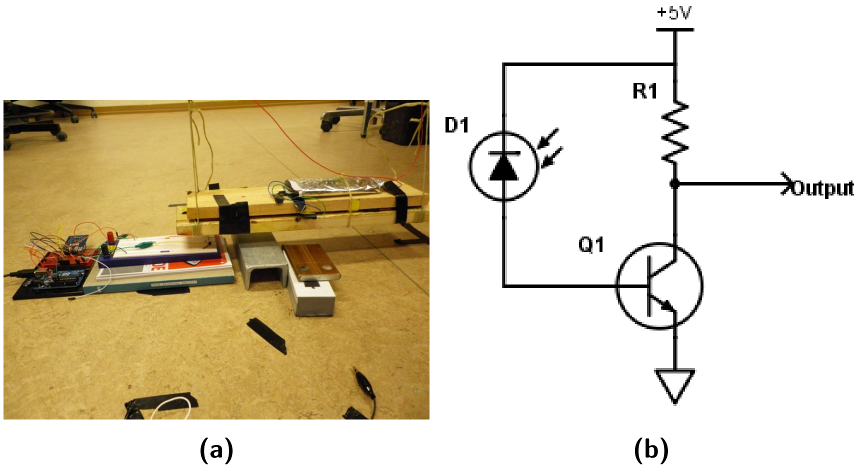


Figure 4.1: 4.1a The setup with the swing without the strong light source lit. To the right in the picture an Arduino Uno and photo detection circuit can be seen and under the swing there is room place the system to be tested. 4.1b Measurement circuit for the photodiode with R1 = 10 $k\Omega$.

4.1.2 Inductive system from the previous thesis work

First off the circuit from the previous master thesis work [Sund, 2014] was tested, which has a resonance frequency of 29 kHz. The detection circuit sent out a 5 V high car detect signal when the voltage over the receiver coil was above a calibratable threshold, implemented using a single comparator. This means that the detection circuit sent out a high signal once every period and that the high signal lasted for the time the voltage over the coil was positive and above this threshold. For testing purposes the threshold was tuned to 2.6 V and the swing was mounted 5.5 cm over the detector. In the original master thesis the detection was made by receiving the signal as an interrupt and once an interrupt had occurred the interrupt handling was turned off for a fixed amount of time, corresponding to the estimated speed the car should have on the road the system was implemented on. One disadvantage with this way of doing a detection is the lack of precision since it is only relying on a correct tuning of a threshold for the field's amplitude. Furthermore the detection has little protection against interference or altering of the received frequency and amplitude. In the tests performed in this work advantage was taken of the fact that the transmitter coil is made in a horizontal loop and the receiver is made in a vertical loop. The envelope of the transmitted magnetic field therefore has the shape of two lobes with a zero point in between. This zero point corresponds to the point straight underneath the transmitter coil, shown as the dotted lines in figure 4.2. Using an Arduino Uno[1] the 30 kHz changes were filtered

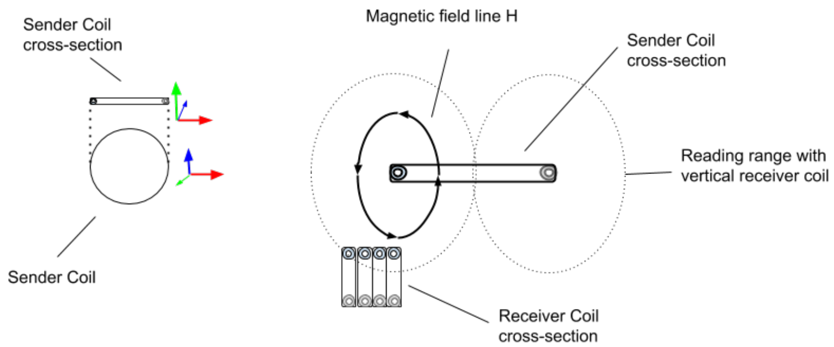


Figure 4.2: The magnetic field sent out from the horizontal coil and picked up by the vertical receiver coil.

out and time stamps for the detected rise and fall of the first and second lobe were gathered. A very precise measurement of the detection distance was made using the speed information and reference time stamps of the photo diode, see figure 4.3. This shows that the low point between the lobes can easily be detected with a

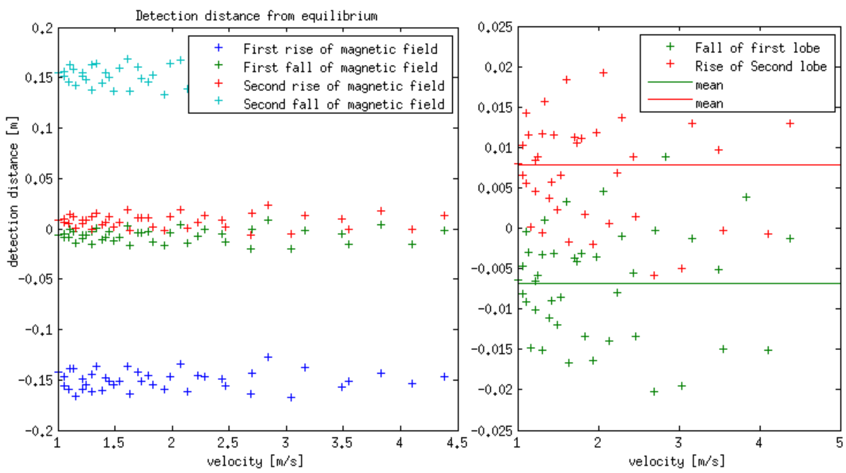


Figure 4.3: Plot of time stamps for rise and fall detection in reference to the photo diode

very high degree of accuracy. In this first experiment the position error was below

[1]open-source prototyping platform, https://www.arduino.cc/

1 cm. The maximum theoretical speed of this method is limited by the rise time of the magnetic field since the magnetic field strength has to rise high enough for a change to be detectable. The point between the two lobes should always exist since the two lobes have a half period phase shift to each other and therefore the signal always has to pass a point with zero magnetic field.

To test the consequences of a phase shift and other possible frequencies, a digital square wave oscillator was programmed on an Arduino Uno. By using the internal clock and timers, a pin toggle of 29.9 kHz with the possibility to turn the signal on and off fast was created. A push-pull transistor output stage was built following the later stage in the schematics for the transmitter in the previous master thesis [Sund, 2014, p10]. The transmitter coil was then powered by this output stage and driven by the Arduino pin toggle signal. Since the transmitter coil acts as a resonance circuit the square wave gets filtered into a sinusoidal magnetic field. A stationary simulation setup was made by placing the transmitter and receiver coil still at the optimal lateral transfer position but keeping the desired height constant, see fig 4.4.

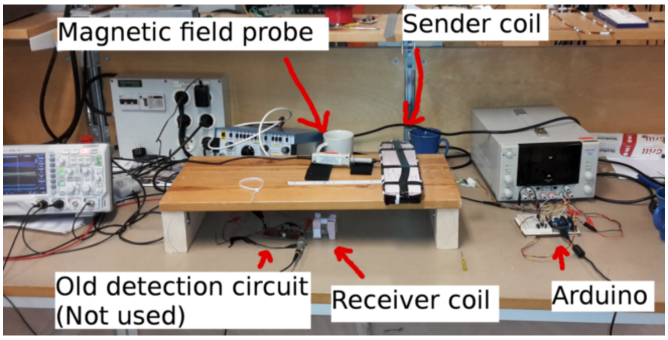


Figure 4.4: The stationary simulation setup. A Oscilloscope measured the magnetic field from transmitter coil, voltage over receiver coil and driving signal from Arduino

A passing car could roughly be simulated by turning the square wave signal to the new output stage on and off for the time corresponding to a distance at a specific speed. Moving the transmitter laterally at a height of 10 cm over the receiver in the movement axis gave an approximation of the strength of the inductive field at different distances from the receiver. The distance from the midpoint where the induced voltage in the receiver was under 10% of the maximum value was around 0.5 cm. This gave a 1 cm long vertical distance, 10 cm under the transmitter, where the magnetic field was almost zero.

4.1.2.1 Simulating high speeds at 29.9 kHz

To simulate a car passing the system at 100km/h, the Arduino was programmed to turn on the signal for 4 ms, turn off the signal for 400 µs, corresponding to 1.1 cm, and then turn on the system again for 4 ms. This resulted in figure 4.5.

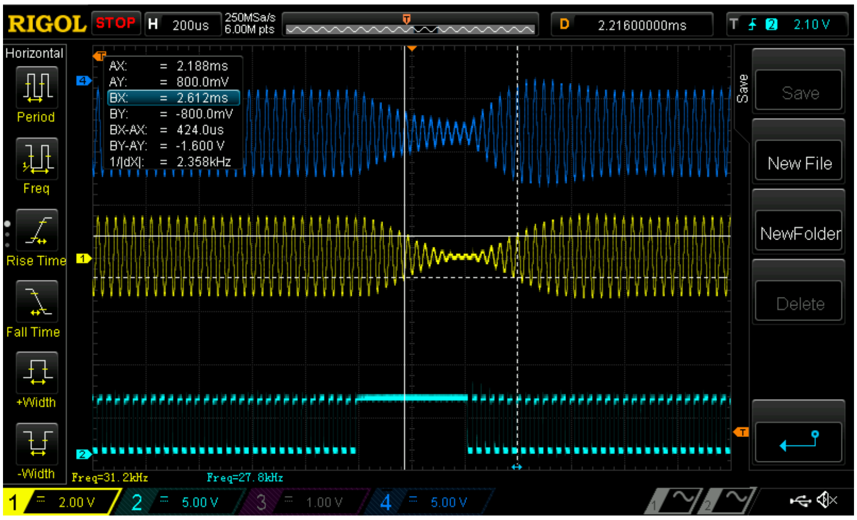


Figure 4.5: A simulated car passing using the 30 kHz receiver and transmitter placed at optimal reading range and then turning off the 30 kHz square wave for 400 µs to simulate a 1.1 cm low magnetic low point that passes in 100 km/h. Blue is the magnetic field radiated from the transmitter. Yellow is the voltage over the receiver. Turquoise is the 5 V square wave generated by the Arduino feed into the transmitter coil.

To simulate the phase shift the off time had to be a factor of the period time. For 29 kHz the period time is 33 µs so an off time (the signal can be high or low) of 33 µs should make a phase shift and force the receiver to pass through a low-point. By experimenting, a delay of 11 µs made the Arduino pause its clock for 15 µs making a 30 µs off time and phase shift shown in figure 4.6.

This shows that it is possible to detect the passing inductive coil at very high speeds by using the fact that the signal makes a phase shift and the magnetic field passes through a low-point. But it should be better to use a higher frequency so there is a longer low-point in relation to the period time to detect. Increasing the frequency gives lower rise times and longer low point for the coils which in turn would give a faster and more distinct detection so the receiver coil and the receiver circuit were remade.

4.1.3 Experimenting with different near field antennas

The transmitter and receiver are both resonating coils acting as near field antennas. The word antennas is used here in a wider sense than the common definition, since we are calling them coil antennas. Equation 3.2 was used to adjust the resonance frequency of the LC circuit. The inductance was chosen such that

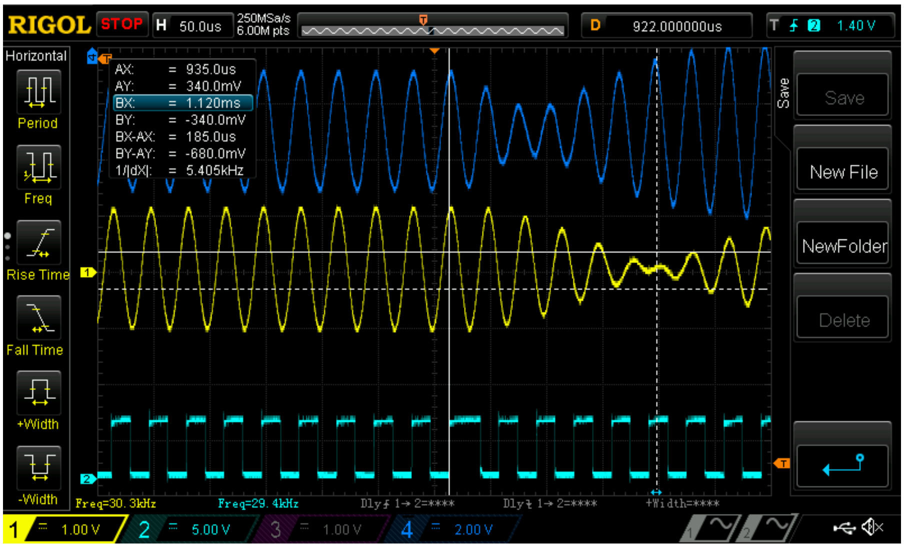


Figure 4.6: A Simulated phase shift using the 30 kHz receiver and transmitter placed at optimal reading range and then pausing the 30 kHz square wave for 15 µs. Blue is the magnetic field radiated from the transmitter. Yellow is the voltage over the receiver. Turquoise is the 5 V square wave generated by the Arduino feed in to the transmitter coil. The phase shift takes place at the time Ax (solid line) and is seen in the receiver coil 185 µs later (dotted line).

simulations of the circuit showed a good rise time and relatively low transients and the capacitance was adjusted according to eq. 3.2. A rough approximation of how many windings and what cross section area a coil should have to get a certain inductance was empirically found.

Copper thread with a diameter of 0.6 mm was used in order to lower the power losses. The inductance of the coil was measured by connecting a known capacitor in parallel with the unknown coil, doing a frequency sweep and looking for the frequency where the voltage over the coil had a peak. The inductance was then calculated using eq. 3.2. The resonance frequency was easier to observe by connecting a big resistor in series with the parallel LC circuit because it lowered the voltage span over the LC circuit making it easier to observe the peak (10 and 100 kΩ were used in the experiments).

In the experiments presented in this report, the dimensions of the transmitter coil antenna tested with were kept roughly the same as in Sund's work but the receiver coil antenna had smaller diameter and a ferrite core. First the new antennas were tuned to 300 kHz for a higher position accuracy since more periods per unit of time would be available for processing. Later on the antennas were retuned to 140 kHz because the maximum allowed field strength at 300 kHz is very low compared to around 9 −148 kHz [PTSFS, 2015, p.13]. And since the chosen RFID system runs on 125 kHz, 140 kHz seemed like a good frequency to aim for. The maximum allowed field strength at 140 kHz and a distance of 10 m from the antenna is 42 dB uA/m [PTSFS, 2015, p.13]. This corresponds to 0.158 nT:

$$\begin{aligned}
&42\,\mathrm{dB}\,\mathrm{\mu A/m} = 20 \cdot \log\left(\frac{H}{1\,\mathrm{\mu A/m}}\right) \\
&\Leftrightarrow H = 125.892\,\mathrm{\mu A/m} \\
&\Leftrightarrow B = \mu_0 \mu_r H = 0.158\,\mathrm{nT}.
\end{aligned}$$

4.1.4 Simulating, testing and building

The 140 kHz system consists of a transmitter stage and a receiver stage. As the transmitter stage an Arduino Uno acts as a signal generator and together with a transmitter antenna the sending stage is to be mounted underneath a car. The receiving stage consists of a receiver antenna, rectifier electronics and an Arduino Due[2] with an Ethernet and SD-card reader shield[3]. The receiver antenna, electronics and Arduino are to be put inside a road segment mock-up with LED lights on it. Instead of delivering power to a passing car, the segment lights up its LED lights when the car passes.

4.1.4.1 Receiver and transmitter antennas

The transmitter coil or antenna was finally constructed by 33 turns of 0.6 mm copper wire in the shape of an rectangle with a length of 375 mm and width of 70 mm. This gave an inductance of 554 µH. The resonance frequency of the transmitter was tuned to 140 kHz with a series capacitance of 2.470 nF. The receiver antenna was constructed by 49 turns of 0.6 mm copper wire wound around a ferrite core from an old AM radio. The inductance of the receiver antenna then became 128.9 µH and got the resonance frequency 140 kHz with 10 nF. The two antennas are shown in figure 4.7. The skin effect had to be considered when measuring the impedance of the transmitter and receiver LC circuits. Therefore the measurements were made with an LCR meter instead of the usual four-point measurements with the multimeter. The skin effect is the tendency of the current density to be distributed in the area close to the surface of the conductor and not in the middle. Higher skin effect means that the current flows through an area smaller than the cross section area of the conductor, thus the resistance is higher. This is due to the eddy currents generated inside the conductor which counteract

[2] Arduino Due is a more powerful developing board based on a 32-bit ARM core microcontroller. More information on https://www.arduino.cc/en/Main/ArduinoBoardDue

[3] Arduino ethernet shield is an extention to the Arduino giving it an ethernet jack and a micro SD-card reader. This effectivly giving the Arduino the means to communicate through network and save information on an micro SD-card. More information can be found on https://www.arduino.cc/en/Main/ArduinoEthernetShield

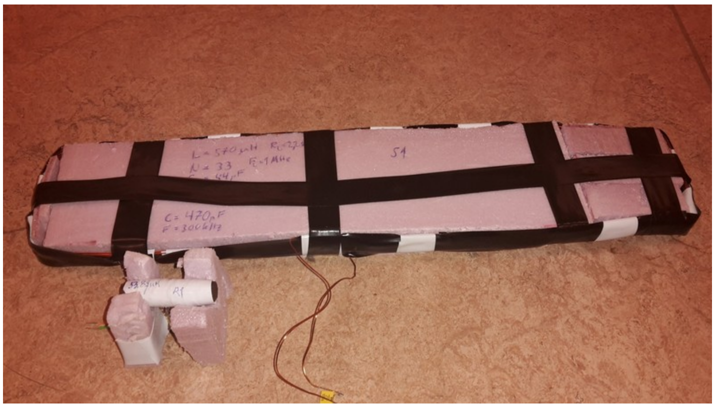


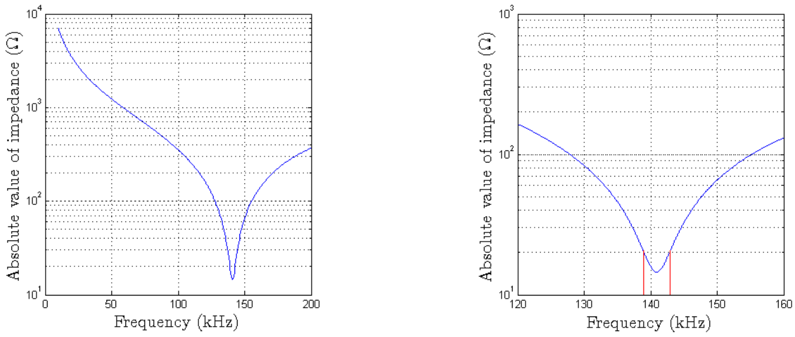
Figure 4.7: The transmitter (top one in picture) and receiver antenna, at an earlier resonance tuning. The dimensions and shape are the same as in the chosen 140 kHz setup. Only the number of windings and the tuning capacitance value differ.

the current in the middle of the conductor. The skin depth in a copper wire can be calculated according to:

$$\delta = \sqrt{\frac{2}{\omega \mu_r \mu_0 \sigma}}$$

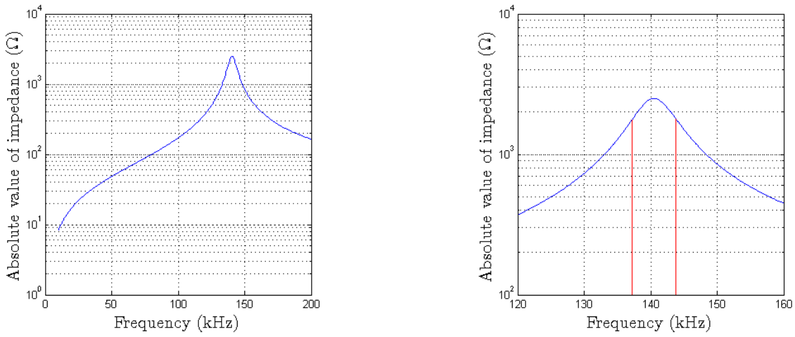
where μ_0 is the magnetic permeability of free space $\mu_0 = 4\pi \cdot 10^{-7} T \cdot m/A$, μ_r is the relative magnetic permeability (it has the value 1 for air), ω is the angular frequency and σ is a parameter specific to copper. The formula shows that the skin effect is higher for higher frequencies and it can be calculated that the skin effect starts affecting the resistance first after 49 kHz where δ is 0.3 mm, i.e equal to the radius of the wire which means the current flows through the whole cross section area. At the frequency of 140 kHz δ is approximately 0.178 mm which means that a little over half of the cross section area of the conductor is used. Since the radius and the skin depth are so close to each other there is no equation to calculate the resistance of the conductor. Another factor probably affecting the resistance in the coil more strongly is the proximity effect, i.e losses in the conductor due to eddy currents induced by magnetic fields that are close by. The series resistance in the transmitter circuit was measured to 14.51 Ω and to 2490 Ω in the parallel circuit at the exact resonance frequency of 140.660 kHz. The quality factors for the transmitter and receiver circuits are 32.64 and 21.98 respectively. The time constants are 73.77 µs for the transmitter and 49.74 µs for the receiver according to eq. 3.7. Impedance measurements for the transmitter antenna and receiver antenna can be seen in figures 4.8 and 4.9.

The reason behind winding the receiver around a ferrite core was to lower the number of turns and the radius while keeping the inductance value high. The reason to want a high inductance value is to get a stronger signal. A side effect of using a ferrite core is that the core might be saturated by static magnetic fields



(a) (b)

Figure 4.8: Bandwidth of the tuned transmitter antenna. The bandwidth is approximately $4\,\mathrm{kHz}$



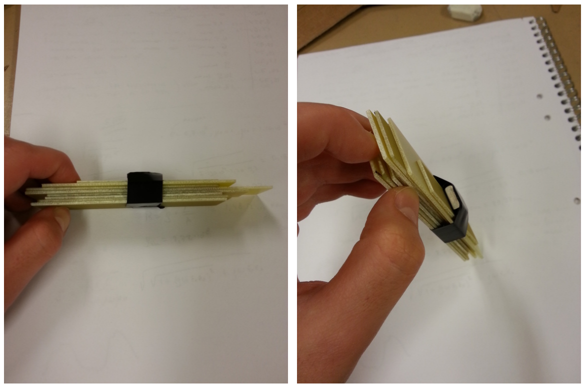
(a) (b)

Figure 4.9: Bandwidth of the tuned receiver antenna. The bandwidth is approximately $6.5\,\mathrm{kHz}$

from the electrical road. To test the affect the current of the road will have on the signal strength in the inductive detection an experiment was made. The aim of the experiment is to see how a neodymium magnet at different distances affects the amplitude of the induced voltage in the receiver antenna.

4.1.4.2 Test of static magnetic impact on ferrite core

To test the effect a strong magnetic field would have on the $140\,\mathrm{kHz}$ induced voltage in the receiver antenna, a neodymium magnet was used and placed at different distances from the receiver. The receiver was placed at optimal reading range from the transmitter. A square wave generated by an Arduino Uno was sent as input to the transmitter and the induced amplitude was measured while the magnet was placed at different heights from the receiver, see figure 4.10.



(a) (b)

Figure 4.10: A piece of neodymium magnet was fastened with tape on slices of hard plastic and placed above the magnetic receiver, the number of slices used gave a measurement of the height between them which was hard to measure otherwise because of the attraction force. Each slice had a thickness of $1.5\,\mathrm{mm}$.

Figure 4.11 shows the results. The peak around the height of 4.5 mm is probably a result of the resonance in the transmitter coil and receiver coil are not fully aligned. When the magnet saturates the ferrite core slightly the permeability is lowered, inductance raised and the resonance frequency lowered in the receiver resonance circuit. This probably result in the receiver and transmitter circuits aligning better and therefore a peak arises. As it can be seen in the figure the receiver is not affected at all if the magnet is placed at a distance higher than approximately 7 mm.

The exact characteristics of the neodymium magnet were not known, but an usual neodymium magnet of grad N50 has a residual flux density, Br, of 1.4 T [NdFeB Specialists E-magnets UK, 2016]. The magnet used is roughly 5 mm in diameter and 2 mm thick. An expression [Magnetics, 2016] describing the magnetic field strength along the central axis of a round neodymium magnet is:

$$B = \frac{Br}{2} \cdot \frac{t+x}{r^2 + (t+x)^2} - \frac{x}{\sqrt{r^2+x^2}},$$

where r is the radius of a cylinder magnet, t the thickness of the magnet and x the distance from the face of the magnet to the measured point. The magnetic field of the magnet at the distance 7 mm using this formula results in roughly 56 mT. According to our experiments in figure 4.11 this is the lower bound that affects the receiver. The static magnetic field at a distance r from a wire carrying a current I is given by:

$$B = \frac{\mu_0 I}{2\pi r},$$

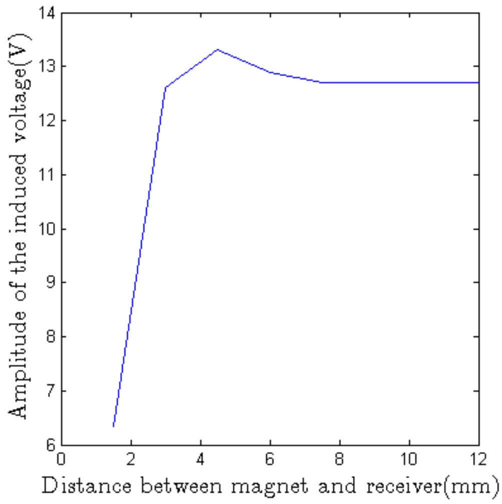


Figure 4.11: How the amplitude of the induced voltage over the receiver varies with a neodymium magnet placed at different heights from it.

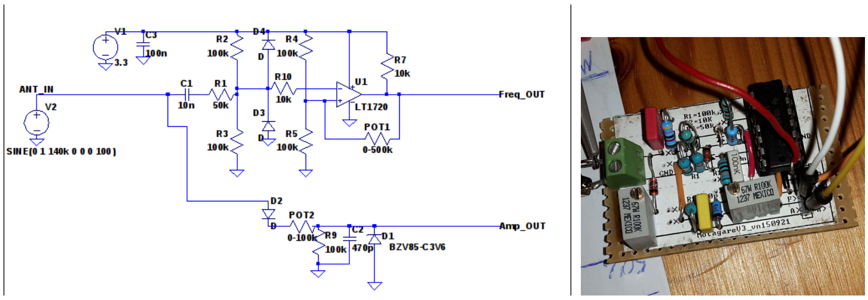
where μ_0 is the magnetic permeability of free space $\mu_0 = 4\pi \cdot 10^{-7} T \cdot m/A$. Assuming that 10 mm is the nearest distance from the receiver the pickups will drain current and that the maximum flowing current will be 400 A. This results in a maximum induced magnetic field of 8 mT at the receiver ferrite core.

The probable magnetic field induced by the wires and pickup is lower then a fifth of the field strength that made an impact on the ferrite core. Therefore the induced field from the high current in the system should not affect the inductive detector.

4.1.4.3 Rectifier and receiver Arduino

Once the resonance frequency was decided upon and antennas built a receiver circuit was made. Due to previous experience with Arduino it was easy to keep using it. The Arduino Due is used as a receiver since it has higher performance than the Uno board with a faster processor, more memory and faster ADC (analog to digital converter).

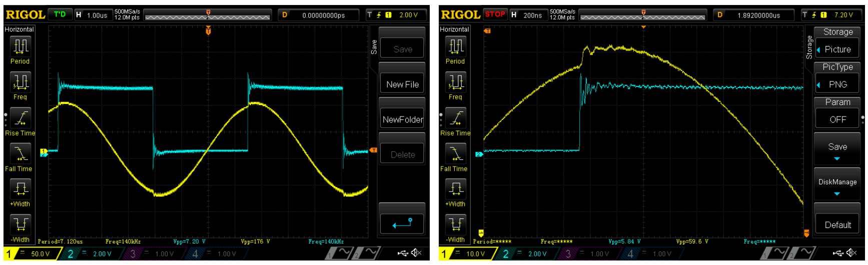
As seen in previous tests, using hardware to send a digital signal on the rise of the wave and hence detect the frequency is one way to detect the signal. A second way is to convert the signal to an analog value corresponding to the amplitude of the waves and then sample it. A third way would be to just sample the raw wave and do all analysis in software. As a first step it was decided to try to do the detection on the signal amplitude and check that the signal had the correct frequency by frequency detection. A sketch of the detection circuit for the receiver is shown in figure 4.12a. This circuit converts the frequency to a digital square wave and the amplitude to an analog value which could be read using the ADC. The Arduino Due can only handle voltages up to 3.3V so both circuits are built to have an output in the 0-3.3V range. The frequency analyzing was however never implemented in code since the bandwidth of the antennas was so narrow and the detection algorithm works satisfactory using only amplitude information. The inductance in the transmitter antenna was near the lower acceptable limit,



(a) **(b)**

Figure 4.12: *4.12a)* Schematic of the receiver antenna electronics with a rectifier in the lower half and in the half above is a comparator converting the sine wave to a square wave signal. *4.12b)* The PCB of the receiver electronics

leading to a suboptimal filtering of the square wave to sine wave shown in figure 4.13. The ripple in the driving square wave was caused by the antenna load. In this figure it is also possible to read out that the resonance frequency of the antenna is not exactly 140 kHz but slightly lower as the sine period is slightly longer than the driving 140 kHz.



(a) **(b)**

Figure 4.13: A none perfect filtering of the square wave to sine wave in the sending antenna. The blue curve is the driving square wave signal. The yellow curve is the field out from the antenna measured with a magnetic-field probe

Doing the same experiment as with the antennas in section 4.1.2.1 a much faster rise time could be seen. In figure 4.14 the reading from the oscilloscope during a stationary test with the signal generating Arduino pausing the signal for 400 µs is seen as a faster result than that of the 30 kHz in figure 4.5.

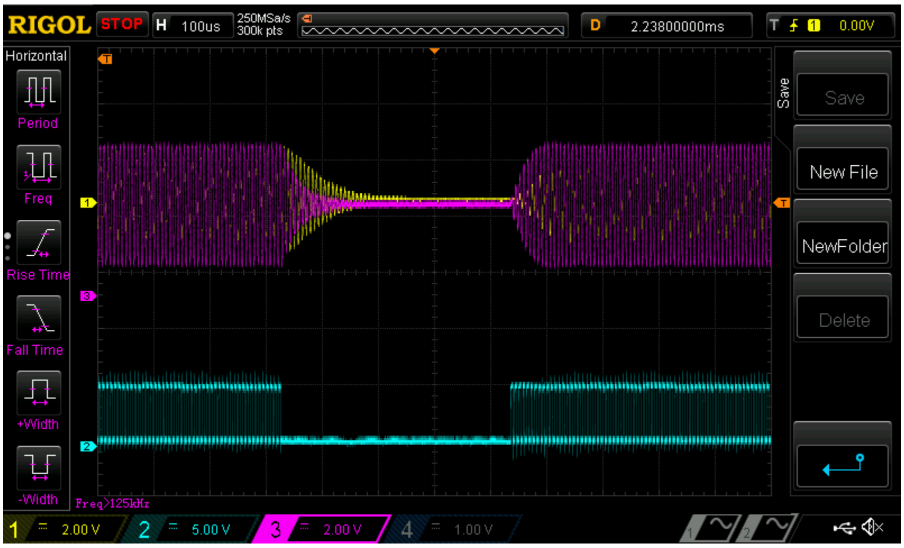


Figure 4.14: Simulated response of the receiver and transmitter (tuned to 140 kHz) at a speed of 100 km/h, i.e the square wave is paused for 400 µs. The transmitter and receiver are placed at optimal reading positions. The pink signal is magnetic field from transmitter coil. The yellow is the voltage over the receiver coil. Blue is the driving square wave signal from the Arduino.

4.1.4.4 Building a mock-up segment

To test the system a mock-up of a road segment was built. The mock-up was equipped with LED strips and lights, instead of power giving rails, so that a visual observation could take place. The driving electronics built for the LED lights are shown as schematics in figure 4.16. The mock-up was constructed by bending a metal plate to a shape resembling the shape of the rail segments. The aim was to make a mock-up resembling the magnetic features of the real theoretic rail segments to make an as close as possible proof of concept test. The mock-up rail was also used to analyze the EMI impact on the rail as further described in section 4.3. The interior of the mock-up consists of batteries to power the LED strips and receiving electronics. To hold everything in place styrofoam, tape and hot-glue were used. The final result of this first mock-up can be seen in the figure 4.15 together with the transmitter antenna in a box to the left.

The driver circuit for the power LED is shown in figure 4.16a. The two transistors work together in regulating the current to the LED in such a way that when the current through M1 is too big the npn transistor is triggered thus reducing the

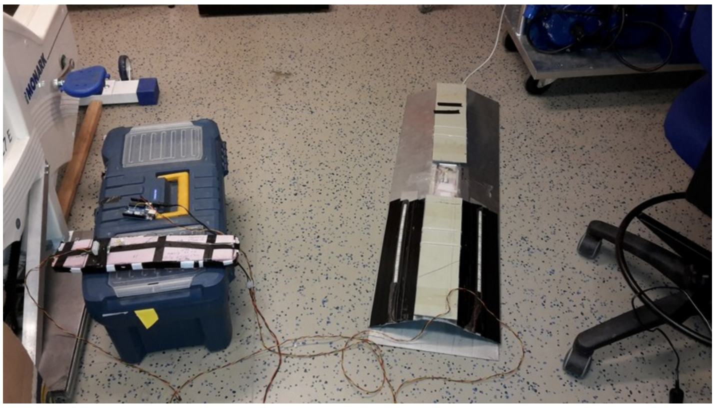
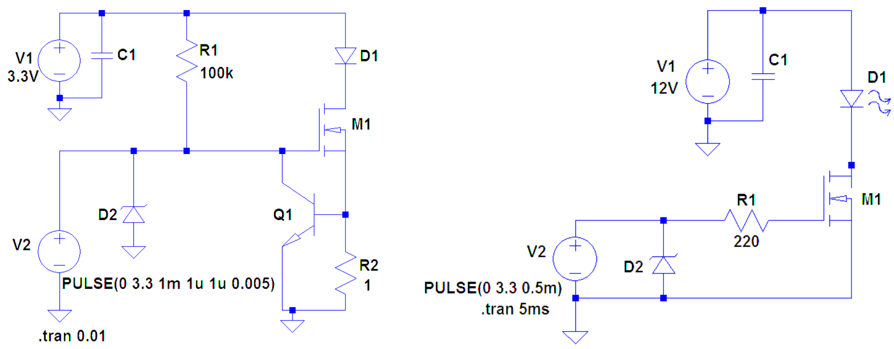


Figure 4.15



(a) **(b)**

Figure 4.16: a) Driver circuit for the power LED, D1 is the power LED, M1 - MOSFET(P80NF10), C1- 1000uF, D2 - 3.3V. b) Driver circuit for the LED strips, D1 is the LED strip, C1- 1000uF, D2 - 3.3V, M1 - MOSFET(P80NF10)

current. The driver circuit for the LED strips is a simple switch configuration, i.e when the Arduino pin is high the transistor starts leading current and the strip is turned on, see figure 4.16b. The Zener diodes have the purpose to protect the Arduino in case something goes wrong and the voltage gets too high but it would have been safer to use a optocoupler instead.

4.1.4.5 Detection algorithm and code

In figure 4.17 an eight states algorithm is shown. In the Arduino code of the road segment the detection algorithm is implemented as function `ampCalcFix()`, see line `764` in apendix A.1. In the algorithm three tests and one timing are made

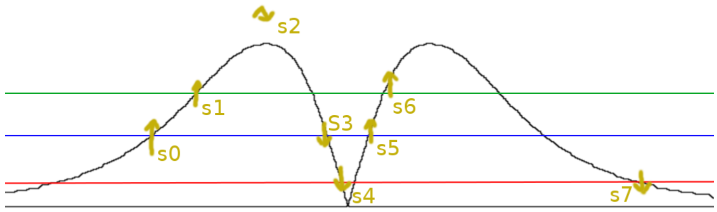
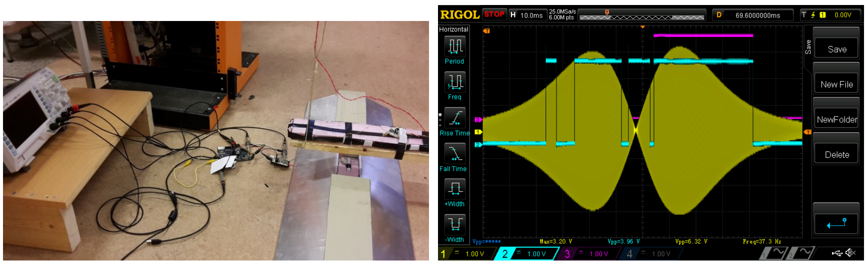


Figure 4.17: Help sketch for eight state detection algorithm

to ensure a correct detection. First the time between s0 and s3 has to be longer than 100 µs (sorting out short spikes). Then the time between s0 and s1 has to be longer than the time between s3 and s4 (steeper slope at zero point between slopes). And finally the time between s0 and s3 has to be longer than the time between s3 and s5 (to not waiting too long for the second lobe). If too much time passes since s0 was passed, then the state machine is restarted. If none of the tests failed, the detection is considered correct and a signal is sent at s6 to turn on the next power rail. In figure 4.18b signals from an indoor test with the swing /pendulum setup are showing that the detection works. The signal used to turn on the LED strips is the pink one shown in figure 4.18b.



(a) **(b)**

Figure 4.18: 4.18a Swing dropped from 1 m Giving a passing speed of around 4 m/s (almost 15 km/h).The Arduino Due is using a state machine with eight states 4.18b. Each state transition makes a change in the light blue (signal 2) curve. And as the state has transitioned in the right way the turn on flag, here seen as pink (signal 3), is turned on and off.

4.1.5 Testing the detection on a road with a car

As the indoor tests showed that the detection worked and stable results were collected it was decided to do a test outdoors with a real car passing over the rail mock-up with the implemented receiver. The described detection algorithm, oscilloscope readings and data dumps of the sampled amplitude values were tested with a real car passing. The test took place in the fuel yard of the combined heat and power plant in Örtofta, where a car could safely accelerate and pass the mock-up segment at high speed. The transmitter antenna and driving Arduino were placed in a box mounted behind a Volvo V70 shown in picture 4.19. The road rail mock-up was placed and filmed from multiple angles with high speed footage filming the LED lights from the side shown in picture 4.20. Multiple passes over

Figure 4.19: The sending circuit mounted on the car

Figure 4.20: The rail mock-up at the Örtofta test location. A high speed camera can be seen rigged in the far left of the picture.

the mock-up were performed and all the passes were correctly detected. Figure 4.21 shows footage from the high speed camera where it can be seen how the LED lights turn on when the car passes over the mock-up at 100 km/h. The transmitter lies in the front of the box and the LED lights are supposed to turn on when the rise of the second lobe is detected at s6 and turn off at the fall of the second lobe shown in the earlier figure 4.18b.

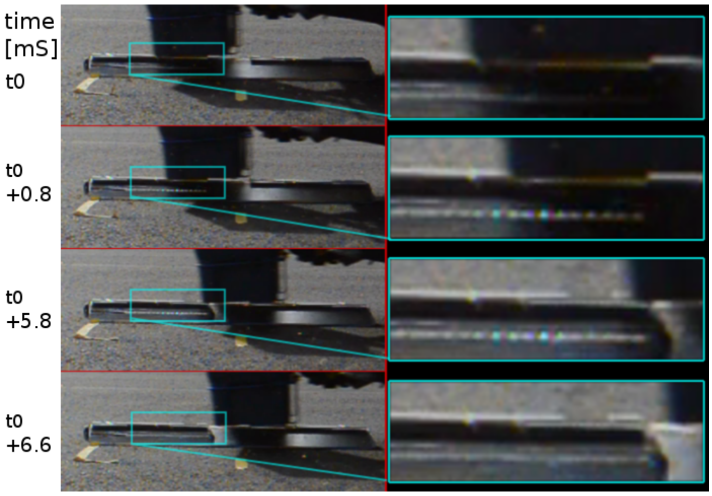
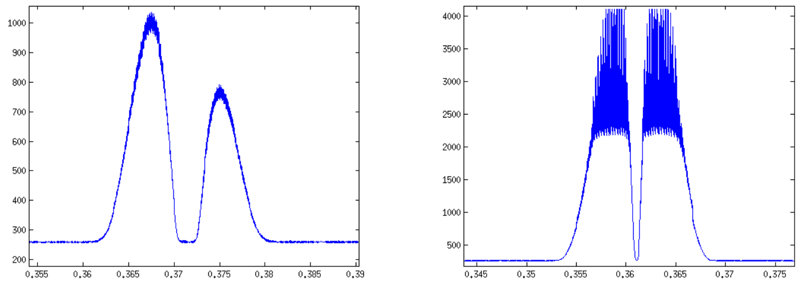


Figure 4.21: Detection of car passing in 100km/h from the left. Screen-shots from a high speed camera to the left and zoomed in on the right. The LED stripe is turned on in the middle two frames.

The sampled amplitude values were also dumped to an SD card by the Arduino. In figure 4.22 two over passes are presented. One when the car passed a little to the side of the mock-up rail and one when the car passed straight over. The amplitude damping of the receiver was not properly tuned and as a result a high ringing is observed at the higher values. This ringings are a weakness in the receiver electronics construction whose exact cause was not fully investigated.



(a) **(b)**

Figure 4.22: Saved values from two different passes by the car. In 4.22a the car passed in an angle to the side of the mock-up. In 4.22b a perfect passing is shown. High ringings are observed at higher amplitudes, caused by a fault in the receiver electronics or the arduino sampling.

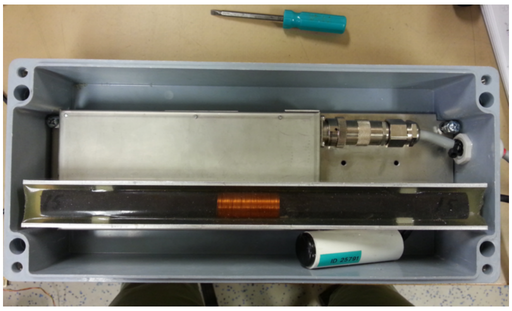
4.2 Car identification

The initial ideas for the identification of a car were to test HF or UHF RFID modules since they had successfully been used in similar applications, see section 3.1.1.3. There were also some ideas about building a radio transceiver from scratch but that idea was quickly discarded since none of the authors had sufficient radio knowledge and the extensive work wouldn't have permitted meeting the deadline for the project. Before deciding which path to continue on, the TSS company was found. Since their products are adapted to traffic environments and they have impressive results the company was contacted in order to borrow their products for testing. Three tags and and one antenna were used in testing, see figure 4.23a and 4.23b.

The recommended distance between the antenna and the tag is 45 cm in order to get good results at high speeds. But tests were made at low speed with a cart, which the antenna is mounted on, driving over the tags in the lab. The tests showed that the tag was successfully read at lower distances at low speed. In order for the tags to not interfere with each other the distance between them should be at least 40 cm.

The antenna communicates with the computer through a serial RS232 connection written in `C`. The main code on the car side, found in appendix A.2.6, is written in `C++` and consists of four threads: one for communication between car and antenna, two threads handling incoming and outgoing messages and one tread that lets the user manually insert the speed.

(a)



(b)

Figure 4.23: a) Picture showing the TSS products used, the antenna and a tag. The antenna and the processing unit are enclosed in the box. The box has a length, width and height of 36x16x9 cm b) The antenna is a coil wound around a core. The processing unit and a small tag for testing purposes are also shown.

4.3 EMI

It is important to investigate the electromagnetic interference under an electrical car driving on a conductive electrical road so that the position and identification electronics can be adjusted accordingly.

Firstly, the behavior of electrical sparks between the road and the pickups was analyzed by using a setup made by one of the project's supervisors, Lars Lindgren. These sparks are generated if the pickups bounce off the road a bit and lose contact with it for a short amount of time which can happen if the car is driving at high speeds. Information about the EMI under a train would have been interesting to compare to, but no solid data could be found. Secondly, tests were made to investigate if the car motors could cause any electromagnetic interference. To test the interference from an electric car the positioning coil antenna was placed on the ground and an electric car was droven over it. The signal from the receiver coil was measured and analyzed.

The test rigg built by Lindgren has been very useful in simulating the behavior of the sparks. The part simulating the road and the pickups consists of two circular metallic tracks with a small sleigh fastened between them. The sleigh has four contacts, two on each track and it is free to rotate, see figure 4.24.

The grey rectangles represent the conducting parts of the rails and the receiver antenna is located in between them. The red loop antenna simulates the path the current takes from the rail and up to the car battery through the pickup. Stray inductances are not shown in the circuit. The capacitor between the loop antenna and ground is added to counteract the effect of all the stray inductances and thus make the induced voltage in the receiver clearer to see.

In the real setup the sleigh is surrounded by a metal case as a safety precaution,

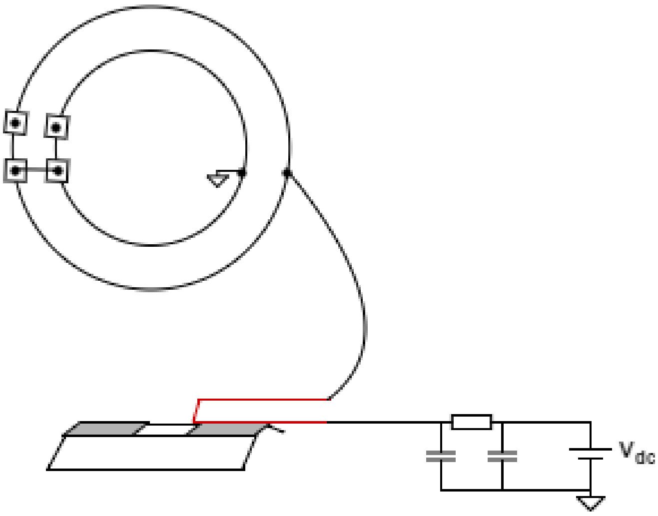


Figure 4.24: The setup for simulating the effect of electrical sparks between the road and the pickups.

see figure 4.25. The whole setup can be seen in figure 4.26. The DC voltage input

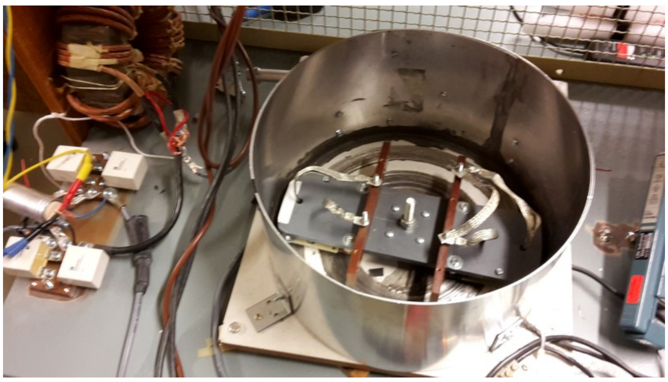


Figure 4.25: Picture of the circular metallic tracks and the sleigh. On the sleigh the four mentioned contact points and short circuiting wires can be seen.

goes to the loop antenna and the sleigh contacts through a load in series with a capacitor, the loop antenna can be seen closer in figure 4.27a.

Two sets of experiments were made. In the first one, the pickup was perfectly aligned above the rail and since the magnetic field in this position is perpendicular to the field in the receiver antenna no disturbances should be registered. In the second one, the pickup was positioned at a 45 degree angle with the rail, shown in figure 4.27b, simulating the situation when the pickup is not centered around the rail but still has contact through the contact shoe. In this case the magnetic field

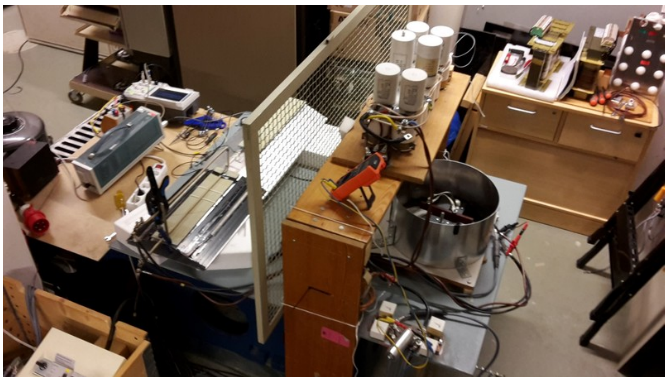
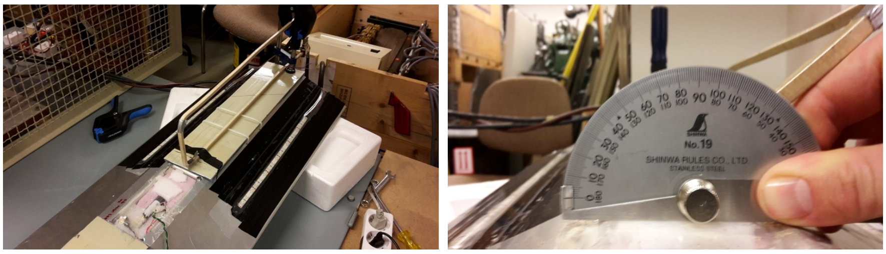


Figure 4.26: The whole setup: in the lower left corner is a wooden box containing transformers and rectifiers with a parallelly connected capacitor to give a stable DC input. On the top shelf is the set of capacitors which have the task of making the ringings sharper and on the leftmost bench is the prototype road.



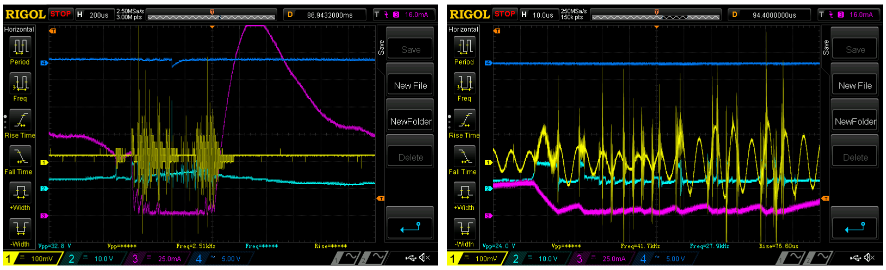
(a) **(b)**

Figure 4.27: On the rail of the prototype there is a loop antenna simulating a pickup. a) In this particular position the pickup is perfectly centered above the rail.b) The pickup has a 45 degrees angle with the rail

in the loop will affect the signal in the receiver antenna. Each experiment was run with both 40 A and 80 A as the input to see how the current difference will affect the receiver. The different signals measured during the experiments can be seen in figure 4.28.

Figure 4.29 and 4.30 show the results when the pickup was placed at 90 and 45 degrees with respect to the road. The amplitude of the ringings doesn't exceed 2 V peak to peak which shouldn't be a problem since our transmitter circuit generates a voltage of approximately 7 V peak to peak in the receiver circuit and the transmitter's magnetic field could be made up to twenty times stronger without exceeding the restrictions that are mentioned in section 4.1.3. The current difference didn't lead to a significant difference in amplitude in the ringings. This is a good result because the relationship between the current increase and amplitude

Figure 4.28: The signals that were measured on the oscilloscope. Yellow is the voltage over the receiver antenna located in between the rails, turquoise and magenta are the voltage and the current through the circular tracks and blue is a signal indicating the rotation speed of the sleigh.

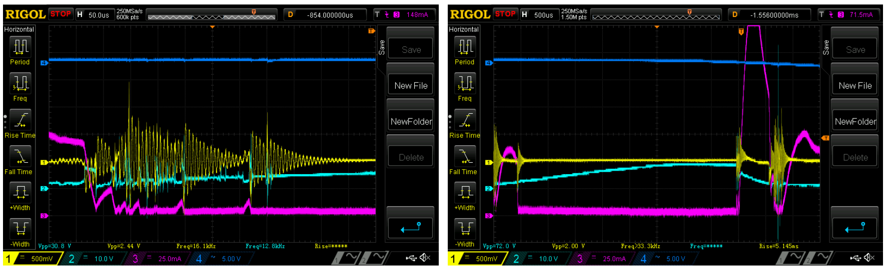


(a) **(b)**

Figure 4.29: The ringings in the receiver antenna when the loop antenna was placed at a 90 degree angle with the rail and the current was 80 A in left picture and 40 A in right picture

of the ringings is not linear. Since the relationship seems to be less than linear it is reasonable to think that the amplitude of the ringings at 400 A will be lower than the voltage our transmitter coil could induce in the receiver antenna.

The reason for the ringings picked up when the antenna was placed at a 90 degree angle can be that the antenna wasn't perfectly aligned. The source of the high voltage spikes was not closely investigated but it is probably related to capacitive and inductive stray fields which can come from cable loops in the circuit or interference from the motor.



(a) (b)

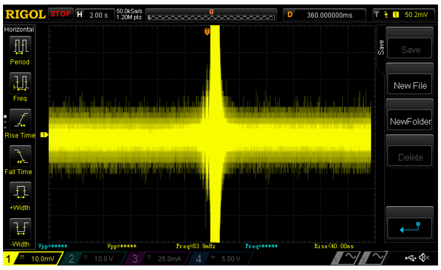
Figure 4.30: The ringings in the receiver antenna when the loop antenna was placed at a 45 degree angle with the rail and the current was 80 A in left picture and 40 A in right picture

4.3.1 EMI from electric car

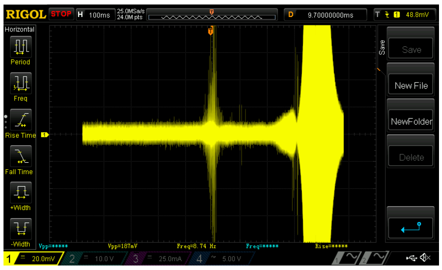
Some small experiments were carried out by taping the transmitter under a Nissan Leaf, a fully electric car, and analyzing the response when it passed over the mock-up rail. The Arduino detection algorithms showed no problems with detecting the transmitter coil passing by. By connecting the receiver antenna directly to an oscilloscope, the possible EMI was intended to be further measured and analyzed. As shown by the two seconds long sample in figure 4.31a, where the car passed the transmitter at low speed, no significant interference was registered around the detection signal. However, an interesting spike could be observed in figure 4.31b. In this passing, a spike could be observed before the transmitter passed the receiver. The transmitter was in this experiment mounted on the rearmost part of the car and the car was driving forward at a speed of about 20 km/h over the mock-up rail. The spike had the same frequency as the transmitter antenna and must have been caused by some part of the car frame forming a loop, that the transmitter induced the frequency in. This car frame loop then acted as an antenna. The amplitude is about a tenth of the amplitude received from the transmitter itself. This experiment shows that no obvious electromagnetic interference seemed to have been caused by the car's power electronics or motors.

4.4 The Complete system

With a working system for detection, a working system for identifying and experiments showing that the EMI would not be a big problem it was decided to put it all together in a complete system to prove the concept. Out of the three communication schemes discussed in section 3.3, scheme one would be easiest to implement. This since the RFID reader is expensive and big but the tags are smaller and easier to mount on the road. Furthermore the communication paths do not have to be deterministic since the timing critical detection is done solely in



(a)



(b)

Figure 4.31: 4.31a A two second long sample of the induced voltage in the receiver when the car was passing over it with the transmitter turned on. The transmitter signal is clearly seen by the receiver. 4.31b The same experiment as in a) but here an extra spike can be seen in the middle of the picture

the Arduino.

As such the idea is that the car reads the RFID tag and sends the read RFID tag ID, an unique car ID and the car speed to the road side unit. The road side unit (RSU) then sends a message to the corresponding rail telling it to be ready to detect an inductive coil passing. From the speed information the RSU also tells the rail how long it should be turned on. Using the inductive detector the rail computer activates the "rail" at the correct time. With the help of a timer the Arduino then turns the rail off again as the time corresponding to the distance and speed have passed. In the complete system, the "rail" is still implemented with LED strips for visualization.

4.4.1 Building

To test the system, one more detection circuit - an antenna for the inductive detector and one Arduino Due, were put together with the same characteristics as the one described in section 4.1.4.3. One more mock-up rail was also built but with a simpler construction. A one meter wooden stud was used as the frame with the antenna and LED strips taped to it. Figure 4.32 shows the complete rail segment with the tags and the LED strips mounted, where the distance between a tag and a receiver coil antenna is 1.15 m. Furthest left in the figure is a tag followed by a one meter negative rail, a receiver antenna and an one meter active, switchable rail simulated by a LED strip. Then again a tag, an one meter negative rail, a receiver antenna and finally a one meter active, switchable rail simulated by a LED strip. Both Arduino rail computers were, as described earlier, equipped with Ethernet shields and connected together with a router and a laptop computer at the side of the road. The laptop computer acted as the road side unit (RSU) in the experiment.

From the time it takes for a car to read an RFID tag and then reach the next in-

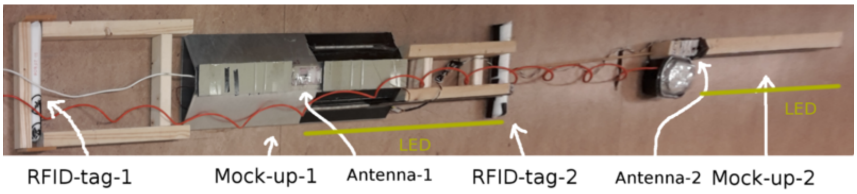


Figure 4.32: The complete rail segment seen from above. LED strips are mounted on the side, shining downwards in the picture. The distance between the center points of a tag and an antenna is 1.15 m. The orange and the white cable in the picture are the Ethernet cables that connect the Arduino computers to the router.

ductive detection point, the tag ID, speed and car identification must have reached the road side computer and activated the upcoming Arduino rail computer. The distance between an RFID tag and the inductive detector is thought to be roughly one meter giving a maximum time of 18 ms if the car is traveling at $200\,\mathrm{km\,h^{-1}}$. Some quick tests of round-trip time for 3G and 4G mobile networks showed way higher latencies. The same tests done on a private, ordinary, 2.4 Ghz wifi showed better results with latencies around 1 ms but with occasional spikes. As such it was decided to use a wifi router to connect the car to the RSU computer for the test. In the car an ordinary laptop was used to communicate with the RFID reader and, using a usb wifi dongle, the RSU.

4.4.2 Programming

The complete program overview showed in figure 4.33 consists of a server with multiple classes written in java. The server can be broken down into three pieces. One part handles the rails (`RSURoad.java` and `RoadConnection- UDP.java`), one part handles the cars (`CarServerThread.java` and `RecThread- .java`) and the last part is a single class implementation which acts as a monitor and makes the communication between the threads 'thread safe' (`Monitor.java`). The flow of the program goes as follows:

1. The car connects to the wifi and the RSU server with an identification of itself through the `CarServerThread.java` .
2. The car reads an RFID tag on the road.
3. The car sends the read tag ID together with the speed to the RSU through the `RecThread.java`.
4. The RSU calculates the corresponding on-time for the received speed in `RSURoad.java` and sends this to the rail computer linked to the tag ID through the `RoadConnectionUDP.java` thread.

5. The rail computer activates the function `ampCalcFix()` and waits for a detection to occur within a time determined by the on-time received. A timer to send a backup message turning off the previous rail is also started.
6. A backup message turning off the previous rail is sent from this rail computer.
7. If a correct detection occurs the LED strip gets turned on by this rail computer. Otherwise if too much time passes, nothing happens.
8. The LED strip gets turned off. Either by this rail computer's turn off timer or by the backup signal sent by the next rail.

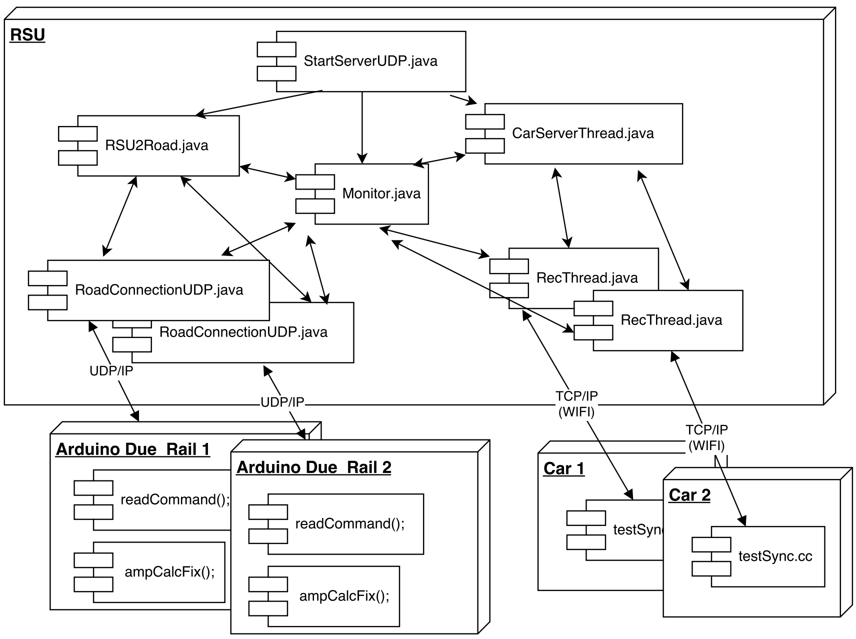


Figure 4.33: The complete program overview. All the classes shown in the RSU except the monitor are threads

4.4.3 Final test

A way to mount the RFID reader and the inductive transmitter antenna on a car was needed in order to perform the final test where the car was supposed to drive over the complete rail segment mock-up at 50 km/h. After some discussion the decision arrived at using a trailer on which to mount the equipment. A trailer was bought to the project and the equipment was mounted as shown in figure 4.34. The equipment was powered using the 12 V outlet in the car. A parking house which was often empty was chosen to be the test place. The equipment was mounted and the complete mock-up was laid out. The test setup is shown in figure 4.35. As described earlier the car needs to send its speed to the RSU when a tag is read for the correct timing to take place. A way to transfer the speed from the car to the laptop used in our system was never implemented. So a speed was instead manually typed into the system and then the driver aimed to pass the road at that speed.

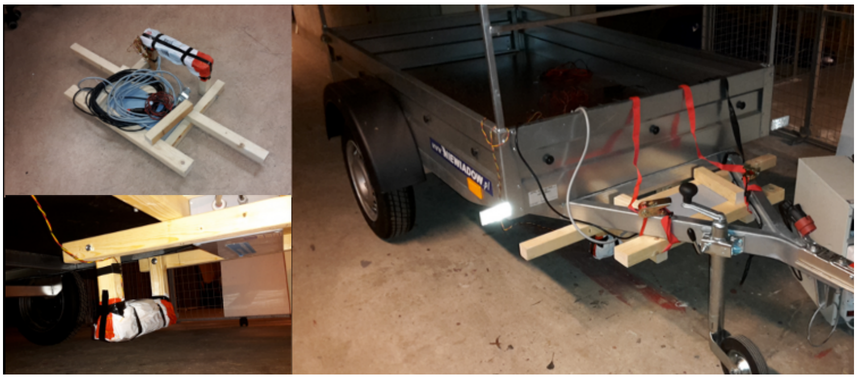


Figure 4.34: In the upper left corner is a picture of the built frame for mounting the reader and transmitter. In the lower left the frame can be seen from the side. To the right is the trailer with the frame, reader and transmitter installed

Figure 4.35: The final test setup is being arranged. The car is going to drive over the mock-up from the left in the picture and the LED strips can be seen pointing towards the camera of this picture as the white stripe on the black tape on the mock-up

4.4.4 Result

A passing of the car driving at the real speed of around 35 km/h and with the system speed set to 33 km/h is shown in figure 4.36 to 4.39. The first picture

4.36 shows the first rail and the LED strip getting activated. Figure 4.37 shows the second rail and LED strip getting activated. In figure 4.38 the first rail and LED strip get turned off again. Finally in figure 4.39 the second rail and LED strip get turned off. Multiple passes were performed, and the system read the tag ID, transferred this information over wifi to the laptop at the side of the road, and activated the detection system in the road segments correctly every time. But since the timing relied on the driver passing the mock-up at the same speed that had been typed into the system, some of the passes did not have the correct turn off timing.

Figure 4.36: The detection and turning on of the first rail segment represented by the LED strip in front of the trailer tires.

Figure 4.37: The detection and turning on of the second rail segment represented by the LED strip in front of the trailer tires.

Figure 4.38: The timing of the first rail-computer (Arduino) turning off the first rail segment represented by the LED strip behind the trailer.

Figure 4.39: The timing of the second rail-computer (Arduino) turning off the second rail segment represented by the LED strip behind the trailer.

Chapter 5

Discussion and future work

The final experiment shows that the software and hardware designed in this thesis works for detecting and identifying a moving vehicle with a little help. By help, it is meant here that the speed has to be manually given to the computer in the car. A possible way to get the speed information from the car into the laptop running our program would be to use an open OBD reader[1] with a suitable API. This was however not investigated due to the limited time frame for the project. The problem of inputting the speed manually, as well as the too small test area and the safety issues, since it was a place open to the public, restricted the posibility of testing the complete setup at full speed. The fact that the RFID system is specified to work at higher speeds, and that the earlier high speed test in section 4.2 went well, still gives some confidence that the complete system would work at higher speeds as well. The final solution built is only a demonstrator for showing a proof of concept of our idea. As such the components and code are not in any way optimized. Software and hardware optimization gives even greater confidence that the system would be robust in tests at higher speeds.

A drawback of the design chosen to work with is the use of an RFID system developed by an external company. This system is built for trains and costs tens of thousands of euros each, making it a very expensive solution. What the price would be at the mass production of the scale road is not known and this could be further investigated in the future.

The EMI experiments in section 4.3 point in the direction that the interference should be possible to overcome and handle. But these experiments are made in a model of how the real system might behave. How the real EMI environment will be, with a 400 Ampere current flowing through the pickups, whose shape, behavior and performance is unknown, is hard to model. It would be good to make further experiments on the model with higher current values to see if the relation between current and interference can be better determined.

[1]OBD stands for "On-board diagnostics" and is a interface connection to a vehicle's self-diagnostic and reporting capability that now is standardized in most cars and often can give real time information

To conclude it is deemed that all of the goals and aims set up for this thesis project were met with a satisfying outcome.

Appendix A

Program Code

A.1 Receiver Arduino Due: The rail road computer

```
#include <SPI.h>
#include <SD.h>
#include <Ethernet.h>

#define DEBUG 1

//metal
//#define RAIL6 1

//tree
#define RAIL7 1

//Ethernet
#ifdef RAIL7
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0x07
};
IPAddress ip(192, 168, 1, 167);
#endif

#ifdef RAIL6
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0x06
};
IPAddress ip(192, 168, 1, 166);
#endif

int port = 5217;
IPAddress server(192, 168, 1, 33);
IPAddress preRail(192, 168, 1, 166);

unsigned int localPort = 8888;
```

```
// An EthernetUDP instance to let us send and receive packets
    over UDP
EthernetUDP Udp;

#define UDP_TX_PACKET 255
#define PRIME 0X0F
#define SETID 0X0A
#define GETID 0x0B
#define TURNON 0X0C
#define TURNOFF 0x0D
#define ACKPRIME 0x06
#define ACKSETID 0X05
#define ACKGETID 0x04
#define ACKTURNON 0X03
#define ACKTURNOFF 0x02
#define PING 0x01

char turnOffMsg[]={'?','O','D','O','1','O','1','K'};

byte lastMsgNbr;
byte tagId[] = {0x00,0x00,0x00,0x00};
byte data[257];
byte dataOut[257];
byte dataLength=0;
byte stage = 0;
byte hexStage=0;
unsigned long etherPreTime=0;
volatile boolean primed=0;

#define LED1 53
#define LED2 51
#define LED3 49
#define LED4 47
#define LED_CAR_PIN 7
#define LED_STRIPE_PIN 5
#define FEQ_PIN 6
#define AMP_PIN A0
#define AMP_HYST 100
#define maxAmpInit 600
#define minAmpInit 1000

#ifdef RAIL6
 //RailInMetal
#define ampTop 1200
#define ampMid 800
#define ampBot 400
#endif

#ifdef RAIL7
 //Rail on tree
#define ampTop 1800
```

```
#define ampMid 1400
#define ampBot 1000
#endif

volatile boolean testLedToggle;
volatile int errorFlag;
File root;
const int chipSelectSD = 4;
char filePrefix[] = "dr5a";
char fileName[16];
int fileNbrH =0;
int fileNbrL =0;
char fileEnd[] = ".txt";

volatile int pasTimerReset;
volatile int sendTurnOff=0;

unsigned long fF,fR,sF,sR;
boolean flagDetect = false; //Detect
boolean flagCollectData=false;
boolean flagCarDetect=false;
volatile int stateCarPassed;
volatile int lastState;
volatile int nextInt;
volatile int nextLow;
#define BUFFER_SIZE  8 //8=8 samples mean value. compered at
    68khz
#define BUFFER_SIZE_BINDEX 3
uint16_t buf[BUFFER_SIZE];

int val;
int maxAmp;
int maxAmpHalf;
int minAmp;
int ampState;
int oldAmpState;
int ampDetectHigh=0;
int ampDetectLow=0;
int ampFailDetect=0;
const int twoFactHighLow = 1; //high > 1^2 * low
#define NBR_STATE 8
unsigned long ampStateTime[NBR_STATE];
unsigned long ampDeltaLowTime;
unsigned long ampDeltaHighTimeDiv;
unsigned long ampStartTime;
unsigned long ampNowTime;
unsigned long ampLastTime;
unsigned long ampDeltaRise;
unsigned long ampDeltaFall;
int ampCarDetect;

```

```
unsigned long timeTest=0;
unsigned long timeTestS=0;

static __inline__ void digitalWriteDirect(int pin, boolean val
    ){
  //http://forum.arduino.cc/index.php?topic=175617.0
  if(val) g_APinDescription[pin].pPort -> PIO_SODR =
      g_APinDescription[pin].ulPin;
  else    g_APinDescription[pin].pPort -> PIO_CODR =
      g_APinDescription[pin].ulPin;
}



void InitsdLogger(){
 /*
  The circuit:
 * SD card attached to SPI bus as follows:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 4
 */
  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelectSD)) {
    errorFlag=1;
    digitalWriteDirect(LED1, LOW);
    digitalWriteDirect(LED4, HIGH);
    // don't do anything more:
    return;
  }
  fileNbrL=0;
  fileNbrH=0;
  sprintf(fileName,"%s%d%s", filePrefix, fileNbrL, fileEnd);
  while(SD.exists(fileName)){
    digitalWriteDirect(LED3, HIGH);
    fileNbrL++;
    if(fileNbrL>7){
      fileNbrH+=10;
      fileNbrL=0;
    }
    int i =fileNbrL+fileNbrH;
    sprintf(fileName,"%s%d%s", filePrefix, (i), fileEnd);

  }
  digitalWriteDirect(LED3, LOW);
}
void ledCount(int count){
  switch (count )
  {
   case 0:
```

```
     digitalWriteDirect(LED4, LOW);
     digitalWriteDirect(LED3, LOW);
     digitalWriteDirect(LED2, LOW);
    break;
    case 1:
     digitalWriteDirect(LED4, HIGH);
     digitalWriteDirect(LED3, LOW);
     digitalWriteDirect(LED2, LOW);
    break;
    case 2:
     digitalWriteDirect(LED4, LOW);
     digitalWriteDirect(LED3, HIGH);
     digitalWriteDirect(LED2, LOW);
    break;
    case 3:
     digitalWriteDirect(LED4, HIGH);
     digitalWriteDirect(LED3, HIGH);
     digitalWriteDirect(LED2, LOW);
    break;
    case 4:
     digitalWriteDirect(LED4, LOW);
     digitalWriteDirect(LED3, LOW);
     digitalWriteDirect(LED2, HIGH);
    break;
     case 5:
     digitalWriteDirect(LED4, HIGH);
     digitalWriteDirect(LED3, LOW);
     digitalWriteDirect(LED2, HIGH);
    break;
     case 6:
     digitalWriteDirect(LED4, LOW);
     digitalWriteDirect(LED3, HIGH);
     digitalWriteDirect(LED2, HIGH);
    break;
     case 7:
     digitalWriteDirect(LED4, HIGH);
     digitalWriteDirect(LED3, HIGH);
     digitalWriteDirect(LED2, HIGH);
    break;
  }
}

void InitTimerFast(Tc *tc, uint32_t channel, IRQn_Type irq, uint32_t time100us)
//http://ko7m.blogspot.se/2015/01/arduino-due-timers-part-1.html
{
    pmc_set_writeprotect(false);
    pmc_enable_periph_clk(irq);
    // clock2 is /8 and clock4 is /128
    TC_Configure(tc, channel, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC
```

```
        | TC_CMR_TCCLKS_TIMER_CLOCK3 );
    uint32_t rc = 263 * time100us; //VARIANT_MCK / 32 / 10000 * time100us;
    TC_SetRC(tc, channel, rc);
    //TC_Start(tc, channel);
    tc->TC_CHANNEL[channel].TC_IER=  TC_IER_CPCS;
    tc->TC_CHANNEL[channel].TC_IDR=~(TC_IER_CPCS);
    NVIC_EnableIRQ(irq);
}
void InitTimerSlow(Tc *tc, uint32_t channel, IRQn_Type irq, uint32_t timeMilli)
//http://ko7m.blogspot.se/2015/01/arduino-due-timers-part-1.html
{
    pmc_set_writeprotect(false);
    pmc_enable_periph_clk(irq);

    TC_Configure(tc, channel, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC | TC_CMR_TCCLKS_TIMER_CLOCK5 );
    uint32_t rc = 32 * timeMilli; //32000 / 1000 * timeMilli;
    TC_SetRC(tc, channel, rc);
    //TC_Start(tc, channel);
    tc->TC_CHANNEL[channel].TC_IER=  TC_IER_CPCS;
    tc->TC_CHANNEL[channel].TC_IDR=~(TC_IER_CPCS);
    NVIC_EnableIRQ(irq);
}
void AdcInit()
// http://nicecircuits.com/playing-with-analog-to-digital-converter-on-arduino-due/
{
  // Setup all registers
  pmc_enable_periph_clk(ID_ADC); // To use peripheral, we must enable clock distributon to it
  adc_init(ADC, SystemCoreClock, ADC_FREQ_MAX, ADC_STARTUP_FAST); // initialize   ADC_FREQ_MIN=20ksample ADC_FREQ_MAX=544ksample
  adc_disable_interrupt(ADC, 0xFFFFFFFF);
  adc_set_resolution(ADC, ADC_10_BITS);
  adc_configure_power_save(ADC, 0, 0); // Disable sleep
  adc_configure_timing(ADC, 0, ADC_SETTLING_TIME_3, 1); // Set timings - standard values
  adc_set_bias_current(ADC, 1); // Bias current - maximum performance over current consumption
  adc_stop_sequencer(ADC); // not using it
  adc_disable_tag(ADC); // it has to do with sequencer, not using it
  adc_disable_ts(ADC); // deisable temperature sensor
  adc_disable_channel_differential_input(ADC, ADC_CHANNEL_7); //pin A0
  adc_configure_trigger(ADC, ADC_TRIG_SW, 1); // triggering from software, freerunning mode
```

```
  adc_disable_all_channel(ADC);
  adc_enable_channel(ADC, ADC_CHANNEL_7); // just one channel
      enabled

  // configure Peripheral DMA

}

// the setup routine runs once when you press reset:
void setup() {
  sendTurnOff=0;
   errorFlag=0;
   pinMode(FEQ_PIN,INPUT);
   pinMode(AMP_PIN,INPUT);
   pinMode(LED_CAR_PIN,OUTPUT);
   pinMode(LED_STRIPE_PIN,OUTPUT);
   //pinMode(13,OUTPUT);

   pinMode(39,OUTPUT);
   digitalWrite(39,LOW);
   testLedToggle=LOW;



   InitTimerSlow(TC2,1, TC7_IRQn, 500); // used for TimerReset
       . last variabel: time [ms]
   InitTimerFast(TC2, 2, TC8_IRQn, 1800); // used for
       TimerNoFreq. last variabel: time [ms/10]
   pasTimerReset=0;


   AdcInit();
   adc_start(ADC);
   pinMode(LED1,OUTPUT);
   pinMode(LED2,OUTPUT);
   pinMode(LED3,OUTPUT);
   pinMode(LED4,OUTPUT);
   digitalWrite(LED_CAR_PIN, LOW);
   digitalWrite(LED_STRIPE_PIN, LOW);
   clearFreqState();
   clearAmpState();
   digitalWriteDirect(LED1, HIGH);

   #ifdef DEBUG
   Serial.begin(115200);
   #endif

   delay(100);
   InitsdLogger();
   PDC_ADC->PERIPH_RPR = (uint32_t) buf; // address of buffer
   PDC_ADC->PERIPH_RCR = BUFFER_SIZE;
```

```
   PDC_ADC->PERIPH_PTCR = PERIPH_PTCR_RXTEN; // enable receive
   delay(1000);
   // start the Ethernet connection:
   Ethernet.begin(mac, ip);
   Udp.begin(localPort);
   lastMsgNbr= 0;

}

void clearFreqState(){
  lastState=-1;
   stateCarPassed=-1;
   nextInt=0;
   nextLow=0;
   fR=fF=sR=sF=0;
}

void clearAmpState(){
  minAmp=minAmpInit;
  maxAmp=maxAmpInit;
  ampState=0;
  oldAmpState=0;
  ampDetectHigh=0;
  ampDetectLow=0;
  ampFailDetect=0;
  digitalWriteDirect(LED4, LOW);
  digitalWriteDirect(LED3, LOW);
  digitalWriteDirect(LED2, LOW);
  //digitalWriteDirect(LED1, LOW);
  for(int i=0; i < NBR_STATE; i++){
    ampStateTime[i]=0;
  }
  ampDeltaLowTime =0;
  ampDeltaHighTimeDiv=0;
  ampStartTime=0;
  ampNowTime=0;
  ampCarDetect=0;
  //digitalWriteDirect(LED_CAR_PIN, LOW);
  //digitalWriteDirect(LED_STRIPE_PIN, LOW);
  ampDeltaRise=0;
  ampDeltaFall=0;
  ledCount(fileNbrL);

}

void startTimerReset(){
  // Start and reset timmer to reset statemachine when no
      change for long
  //uses timmer TC2 chanel 1
  TC2->TC_CHANNEL[1].TC_CCR &= ~TC_CCR_CLKDIS;
  TC2->TC_CHANNEL[1].TC_CCR |=
```

```
  TC_CCR_CLKEN | // enables the clock if CLKDIS is not 1
  TC_CCR_SWTRG; // the counter is reset and the clock is started
  TC_GetStatus(TC2, 1);
  digitalWriteDirect(LED1, LOW);

}

void startTurnOffTimer(){
  //uses timmer TC2 chanel 2
  TC2->TC_CHANNEL[2].TC_CCR &= ~TC_CCR_CLKDIS;
  TC2->TC_CHANNEL[2].TC_CCR |=
  TC_CCR_CLKEN | // enables the clock if CLKDIS is not 1
  TC_CCR_SWTRG; // the counter is reset and the clock is started
  TC_GetStatus(TC2, 2);
}
void startTimerSendTurnOffBack(){
  //uses timmer TC2 chanel 0
  TC2->TC_CHANNEL[0].TC_CCR &= ~TC_CCR_CLKDIS;
  TC2->TC_CHANNEL[0].TC_CCR |=
  TC_CCR_CLKEN | // enables the clock if CLKDIS is not 1
  TC_CCR_SWTRG; // the counter is reset and the clock is started
  TC_GetStatus(TC2, 0);
}

// the loop routine runs over and over again forever:
void loop() {
  ethernet();

  if(primed){ // start recive if primed
    if((adc_get_status(ADC) & ADC_ISR_ENDRX) != 0){ // waiting for buffer to fill upp (544ksample/16=18khz
     ampCalcFix();
     PDC_ADC->PERIPH_RPR = (uint32_t) buf; // address of buffer
     PDC_ADC->PERIPH_RCR = BUFFER_SIZE;
     PDC_ADC->PERIPH_PTCR = PERIPH_PTCR_RXTEN; // enable receive
     }
  }
  //ampPrint();
  if(pasTimerReset){
   clearAmpState();
    pasTimerReset=0;
    primed=0;
  }
  readCommand();
  if(sendTurnOff){
    sendTurnOff=0;
```

```
    sendMsg(preRail,turnOffMsg);
    #ifdef DEBUG
        Serial.print("TurnOff sent");
    #endif
  }
}
void ethernet(){
  char packetBuffer[UDP_TX_PACKET]={};
  noInterrupts();
  int packetSize = Udp.parsePacket();
  interrupts();
  int startIndex=0;
  if (packetSize)
  {
    IPAddress remote = Udp.remoteIP();
    Udp.read(packetBuffer, UDP_TX_PACKET);
    char checksum = 0;
    for(int i=0; i < packetSize; i++){
      checksum ^= packetBuffer[i];
      if(packetBuffer[i]=='?'){
        startIndex=i;
         #ifdef DEBUG
        Serial.print("StartIndex: ");
         Serial.println(startIndex);
        #endif
      }
    }
    if(packetBuffer[0]=='?'){
      if(!checksum){
        data[0] = hexDecode(packetBuffer[1]) <<4; // *16
        data[0] += hexDecode(packetBuffer[2]);
        if((data[0]==lastMsgNbr) && ((data[0] & 0xF0)!= 0)){
          #ifdef DEBUG
          Serial.print("same: ");
          #endif
        }else{
          data[1] = hexDecode(packetBuffer[3]) <<4; // *16
          data[1] += hexDecode(packetBuffer[4]);
          for(int i = 0; i < (data[1]<<1);i+=2){
             #ifdef DEBUG
               Serial.print("index:");
               Serial.println(i);
             #endif
             data[2+(i>>1)] = hexDecode(packetBuffer[5+i])
                 <<4; // *16
             data[2+(i>>1)] += hexDecode(packetBuffer[6+i]);
          }
          if(packetBuffer[(4+data[1])<<1]){
          #ifdef DEBUG
          Serial.println("correct!");
          #endif
```

```
            }
            instruction();
            lastMsgNbr=packetBuffer[1];
          }
          #ifdef DEBUG
            Serial.println("awnser");
          #endif
          Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
          Udp.write('!');
          checksum='!';
          for(int i = 0; i < dataOut[1] + 2; i++)
          {
            #ifdef DEBUG
            Serial.print((char)hexEncode(dataOut[i] >>4)); //
                /16
            Serial.print((char)hexEncode(dataOut[i] & 0xF)); //
                %16
            #endif
            checksum^=hexEncode(dataOut[i] >>4);
            checksum^=hexEncode(dataOut[i] & 0x0F);
            Udp.write(hexEncode(dataOut[i] >>4)); // /16
            Udp.write(hexEncode(dataOut[i] & 0x0F)); // %16
          }

          Udp.write(checksum);
          Udp.endPacket();
          //lastInstruction = millis();
          #ifdef DEBUG
            Serial.println("done");
          #endif
      }
    }else if(packetBuffer[0]=='!'){
      //TODO send again if this is not recived after sendMsg.
        #ifdef DEBUG
            Serial.println("gotAwnser");
          #endif
    }
  }


}
byte hexDecode(byte c)
{
  if(c >= '0' && c <= '9')
  {
    return c - '0';
  }
  else if(c >= 'a' && c <= 'f')
  {
    return c - 'a' + 10;
  }
```

```
  else if(c >= 'A' && c <= 'F')
  {
    return c - 'A' + 10;
  }
  else
  {
    return 0;
  }
}

byte hexEncode(byte n, boolean cap)
{
  if(n >= 0 && n <= 9)
  {
    return n + '0';
  }
  else if(n >= 10 && n <= 15)
  {
    if(cap)
    {
      return n - 10 + 'A';
    }
    else
    {
      return n - 10 + 'a';
    }
  }
  else
  {
    return '0';
  }
}

byte hexEncode(byte n)
{
  return hexEncode(n, true);
}
void instruction(){
   #ifdef DEBUG

        Serial.print("msg recived. lenght");
        Serial.println(data[1]);
         for(int i = 0; i < data[1] + 2; i++)
        {
          Serial.print((char)hexEncode(data[i] >>4)); // /16
          Serial.print((char)hexEncode(data[i] & 0xF)); // %16
        }
        Serial.println("\n msg end");
   #endif

       if(data[0]==SETID){
```

```
        if(data[1]==4){
           #ifdef DEBUG
           Serial.println("setId start");
            #endif
            for(int i=0;i<data[1]; i++){
              tagId[i]=data[i+2];
               #ifdef DEBUG
               Serial.println(tagId[i]);
               #endif
            }
            dataOut[0]=ACKSETID;
            dataOut[1]=1;
            dataOut[2]=PRIME;
            #ifdef DEBUG
            Serial.println("setId done");
            #endif
        }
      }
      if(data[0]==GETID){
        dataOut[0]=ACKGETID;
        dataOut[1]=4;
        for(int i=0;i<dataOut[1]; i++){
            dataOut[i+2]=tagId[i];
            #ifdef DEBUG
            Serial.println(dataOut[i+2]);
            #endif
        }
         #ifdef DEBUG
            Serial.println("getId done");
         #endif

      }
      if(data[0]==PRIME){
        primed=1;
        dataOut[0]=ACKPRIME;
        #ifdef DEBUG
          Serial.println("priming");
        #endif
        if(data[1]==2){
          int onTime = ((data[2] & 0XFF) << 8) + ((data[3])
              & 0XFF);
          #ifdef DEBUG
            Serial.println(onTime);
          #endif
          InitTimerFast(TC2, 2, TC8_IRQn, onTime); // used
              for TimerTurnoff. last variabel: time [ms/10]
          onTime=onTime>>3;
          InitTimerSlow(TC2,1, TC7_IRQn, onTime); // used
              for TimerReset. last variabel: time [ms]  >>3
              makes wait time: 10/8=1. times longer
          startTimerReset();
```

```
              dataOut[1]=1;
              dataOut[2]=0;
              InitTimerFast(TC2, 0, TC6_IRQn, onTime); // used
                  for TimerTurnoff. last variabel: time [ms/10]
               #ifdef DEBUG
               Serial.println("time to send turnoff");
                Serial.println(onTime);
              #endif
            }
          }
          if(data[0]==TURNON){
            #ifdef DEBUG
              Serial.println("turning on");
            #endif
            digitalWriteDirect(LED_CAR_PIN,HIGH);
            digitalWriteDirect( LED_STRIPE_PIN,HIGH);
            dataOut[0]=ACKTURNON;
            dataOut[1]=1;
            dataOut[2]=TURNON;
          }
          if(data[0]==TURNOFF){
            #ifdef DEBUG
              Serial.println("turning off");
            #endif
            dataOut[0]=ACKTURNOFF;
            dataOut[1]=0x01;
            dataOut[2]=TURNOFF;
            digitalWriteDirect(LED_CAR_PIN,LOW);
            digitalWriteDirect( LED_STRIPE_PIN,LOW);
          }


}

void sendMsg(IPAddress rec, char msg[]){
    Udp.beginPacket(rec, localPort);
    Udp.write(msg,8);
    Udp.endPacket();
}

void readCommand(){
  if (Serial.available() > 0) {
    int command= Serial.parseInt();
    if(command==0){
      Serial.println("0=help, 1=list, 2=readCurentFile, 3=
          chose file to read, 4=eraseAll");
    }else if(command==1){
      char tempName[16];
      Serial.println("------print list-------");
      int i = 0;
      int ii= 0;
```

```
      sprintf(tempName,"%s%d%s", filePrefix, i, fileEnd);
      while(SD.exists(tempName)){
        Serial.print("    ");
        Serial.println(tempName);
        i++;
        if(i>7){
          ii+=10;
        }
        int j= ii+i;
        sprintf(tempName,"%s%d%s", filePrefix, j, fileEnd);
      }
      Serial.println("------Done!-------\n\n");
    }else if(command==2){
      File tempFile = SD.open(fileName);
      if (tempFile) {
        Serial.println("----------------------");
        Serial.println(fileName);

        // read from the file until there's nothing else in it
            :
        while (tempFile.available()) {
          Serial.write(tempFile.read());
        }
        // close the file:
        tempFile.close();
        Serial.println("----------------------\n\n");
      } else {
        // if the file didn't open, print an error:
        Serial.println("error opening file");
      }

    }else if(command==3){
      char tempName[16];
      Serial.println("Read file number:");
      while(Serial.available()==0){
      }
      int i = Serial.parseInt();
      if(i==0){
        return;
      }
      sprintf(tempName,"%s%d%s", filePrefix, i, fileEnd);
      if(SD.exists(tempName)){
        File tempFile = SD.open(tempName);
        if (tempFile) {
          Serial.println("----------------------");
          Serial.println(tempName);

          // read from the file until there's nothing else in
              it:
          while (tempFile.available()) {

```

```
          Serial.write(tempFile.read());
        }
        // close the file:
        tempFile.close();
        Serial.println("----------------------\n\n");
      } else {
        // if the file didn't open, print an error:
        Serial.println("error opening file");
      }
    }else{
       Serial.println("No file with that name exists");
    }


  }else if(command==4){
    char tempName[16];
    Serial.println("remove All? (y/n)");
    while(Serial.available()==0){
    }
    int temp = Serial.read();
    Serial.println(temp);
    if(temp != 'y'){
       Serial.println("Aborted");
      return;
    }int i = 0;
    int ii= 0;
    sprintf(tempName,"%s%d%s", filePrefix, i, fileEnd);
    while(SD.exists(tempName)){
      SD.remove(tempName);
      i++;
      if(i>7){
        ii+=10;
      }
      int j= ii+i;
      sprintf(tempName,"%s%d%s", filePrefix, j, fileEnd);
    }
    Serial.println("Done removing");
  }
 }
}



boolean sdLog(char dataString[]){
  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  File dataFile = SD.open(fileName, FILE_WRITE);

  // if the file is available, write to it:
  if (dataFile) {
```

```
    dataFile.println(dataString);
    dataFile.close();
    return 1;
  }
  // if the file isn't open, pop up an error:
  else {
    digitalWriteDirect(LED1, LOW);
    return 0;
  }

}
void ampCalcFix(){
  val=0;
  for (int i=0; i<BUFFER_SIZE; i++){
      val+=buf[i];
  }
  val=val>>BUFFER_SIZE_BINDEX;
      if(val<minAmp){
        minAmp=val;
      }
      if(maxAmp<val){
        maxAmp=val;
      }
      // /
      if((ampState==0)&& (ampMid<val)){
         #ifdef DEBUG
        Serial.println("StartDetect");
         //Serial.println(startIndex);
 #endif
        maxAmp=val;
        ampStartTime= micros();
        ampStateTime[ampState]=ampStartTime;
        startTimerReset();
        ampState=1;
        minAmp=val; //to find new minAmp
        digitalWriteDirect(LED4, HIGH);
        digitalWriteDirect(LED3, LOW);
        digitalWriteDirect(LED2, LOW);
      // /
      }else if((ampState==1)&& (ampTop<val)){
        ampNowTime = micros();
        ampStateTime[ampState] = ampNowTime;
        ampState=2;
        ampDeltaRise = ampNowTime-ampStartTime;
        digitalWriteDirect(LED4, LOW);
        digitalWriteDirect(LED3, HIGH);
        //digitalWriteDirect(LED2, LOW);
      // ^
      }else if(ampState==2 && (val+AMP_HYST) < maxAmp){
        ampDetectHigh=maxAmp;
        maxAmpHalf= maxAmp>>2;
```

```
          ampStateTime[ampState] = micros();
          ampState=3;
          digitalWriteDirect(LED4, HIGH);
          //digitalWriteDirect(LED3, HIGH);
          //digitalWriteDirect(LED2, LOW);
        // \
        }else if( ampState==3  && (val<ampMid)){
          ampNowTime= micros();
          ampLastTime=ampNowTime;
          ampStateTime[3] = ampNowTime;
          ampDeltaHighTimeDiv = (ampNowTime-ampStartTime);//>>
              twoFactHighLow;
//FailDetect 1 To short High!
          if(ampDeltaHighTimeDiv <100){
            ampFailDetect=1;
            ampState=20;
            return;
          }
          ampState=4;
          //ampDeltaHighTimeDiv=ampDeltaHighTimeDiv*10; //fix
          digitalWriteDirect(LED4, LOW);
          digitalWriteDirect(LED3, LOW);
          digitalWriteDirect(LED2, HIGH);
        // \+
//        }else if(( ampState==3 || ampState==4) && val <
    maxAmpHalf){
//          ampStateTime[4] =  micros();
//          ampState=5;
//          digitalWriteDirect(LED4, HIGH);
//          //digitalWriteDirect(LED3, LOW);
//          //digitalWriteDirect(LED2, HIGH);
//        // v
        }else if((ampState==4)&& (val<ampBot)){
          ampNowTime= micros();
          ampStateTime[ampState] = ampNowTime;
          ampDeltaFall = ampNowTime-ampLastTime;
          ampState=5;
          digitalWriteDirect(LED4, HIGH);
          digitalWriteDirect(LED3, LOW);
          //digitalWriteDirect(LED2, HIGH);
//Faildetect 2 to long DeltaFall
          if(ampDeltaRise < ampDeltaFall){
            ampState=20;
            ampFailDetect=2;
          }

        }else if(ampState==5 && (ampMid<val)){
          ampNowTime=micros();
          ampStateTime[ampState]=ampNowTime;
          ampDeltaLowTime = (ampNowTime-ampLastTime);
          ampDetectLow=minAmp;
```

```
        ampState=6;

        digitalWriteDirect(LED4, LOW);
        digitalWriteDirect(LED3, HIGH);
      }else if(ampState==5){
//Faildetect 3 to long low
        unsigned long currentMillis = micros();
        if ((currentMillis - ampNowTime) >=
            ampDeltaHighTimeDiv){
          ampState=20;
          ampFailDetect=3;
          ampStateTime[ampState]= currentMillis;
        }
      // /
      }else if(ampState==6 && ampTop<val){
//Correct Starting rail.
        if(ampDeltaLowTime < ampDeltaHighTimeDiv){
           digitalWriteDirect(LED_CAR_PIN, HIGH);
           digitalWriteDirect(LED_STRIPE_PIN, HIGH);
           ampCarDetect=1;
           startTurnOffTimer();
           startTimerSendTurnOffBack();
           #ifdef DEBUG
            Serial.println("detect");
            //Serial.println(startIndex);
           #endif
        }
        ampState=7;
        digitalWriteDirect(LED4, HIGH);
        ampStateTime[6]=micros();
        //digitalWriteDirect(LED3, HIGH);
        //digitalWriteDirect(LED2, HIGH);

      }else if((ampState==7) && (val<ampBot)){
        ampStateTime[ampState] = micros();
        ampState=0;
        primed=0;
        //digitalWriteDirect(LED_STRIPE_PIN, LOW);
        //digitalWriteDirect(LED_CAR_PIN, LOW);
        digitalWriteDirect(LED4, LOW);
        digitalWriteDirect(LED3, LOW);
        digitalWriteDirect(LED2, LOW);
      }

}
void ampPrint(){
  if(ampState==20){
    char dataString[512] = "";
    sprintf(dataString,"Fail %d! at System time s0: %u uS
        Lowest detect: %d Highest detect: %d", ampFailDetect,
        ampStateTime[0], minAmp, maxAmp);
```

```
    if(ampFailDetect==2)
     sprintf(dataString,"%s dRise: %u, dFall: %u", dataString,
          ampDeltaRise,ampDeltaFall);
    for (int i = 1; i < NBR_STATE; i++) {
        sprintf(dataString,"%s, s%d-s0: %u uS", dataString, i
            , (ampStateTime[i]-ampStateTime[0]));
      }

      //Serial.println(dataString);
      if(!sdLog(dataString)){
        //Serial.println("fail to write 2");
      }
    clearAmpState();
  }else  if(pasTimerReset){
    char dataString[256] = "";
    if((ampCarDetect)){
      //#define NBR_STATE 4
      //unsigned long ampStateTime[4];
      if(ampCarDetect){
        sprintf(dataString, "Correct!");
      }
      sprintf(dataString,"%s Full Detect at System time s0: %u
           uS, dRise: %u, dFall: %u", dataString, ampStateTime
          [0],ampDeltaRise,ampDeltaFall);
      for (int i = 1; i < NBR_STATE; i++) {
        sprintf(dataString,"%s, s%d-s0: %u uS", dataString, i
            , (ampStateTime[i]-ampStateTime[0]));
      }
      sprintf(dataString,"%s, time low between loobes: %u uS,
          lobMax: %d [mV], min: %d [mV]", dataString,
          ampDeltaLowTime, ampDetectHigh, ampDetectLow);
      //Serial.println(dataString);
      if(!sdLog(dataString)){
        //Serial.println("fail to write 1");
      }
    }else{
      sprintf(dataString,"Fail! detect at System time s0: %u
          uS Lowest detect: %d Highest detect: %d",
          ampStateTime[0], minAmp, maxAmp);
      for (int i = 1; i < NBR_STATE; i++) {
        sprintf(dataString,"%s, s%d-s0: %u uS", dataString, i
            , (ampStateTime[i]-ampStateTime[0]));
      }
      //Serial.println(dataString);
      if(!sdLog(dataString)){
        //Serial.println("fail to write 2");
      }
    }
    clearAmpState();
    pasTimerReset=0;
    primed=0;
```

```
  }
}

//TC1 ch 0
void TC8_Handler()
{

 TC2->TC_CHANNEL[2].TC_CCR &= ~TC_CCR_CLKEN;
 TC2->TC_CHANNEL[2].TC_CCR |= TC_CCR_CLKDIS; //disable clock
 //TC_Stop(TC2, 2);
 TC_GetStatus(TC2, 2);
 digitalWriteDirect(LED_CAR_PIN,LOW);
 digitalWriteDirect( LED_STRIPE_PIN,LOW);
  #ifdef DEBUG
        Serial.print("TC8 timeout");
         //Serial.println(startIndex);
 #endif
}
//send turn off to previus rail
void TC6_Handler()
{

 TC2->TC_CHANNEL[0].TC_CCR &= ~TC_CCR_CLKEN;
 TC2->TC_CHANNEL[0].TC_CCR |= TC_CCR_CLKDIS; //disable clock
 //TC_Stop(TC2, 2);
 TC_GetStatus(TC2, 0);
  #ifdef DEBUG
        Serial.print("SendingTurnOff");
         //Serial.println(startIndex);
 #endif
 sendTurnOff=1;
 //sendMsg(preRail,turnOffMsg)
}

void TC7_Handler(){
  TC2->TC_CHANNEL[1].TC_CCR &= ~TC_CCR_CLKEN;
  TC2->TC_CHANNEL[1].TC_CCR |= TC_CCR_CLKDIS; //disable clock
 //TC_Stop(TC2, 2);
 TC_GetStatus(TC2, 1);
 stateCarPassed=10;
 pasTimerReset=1;
 //digitalWriteDirect(LED2, LOW);
 digitalWriteDirect(LED_CAR_PIN,LOW);
 digitalWriteDirect( LED_STRIPE_PIN,LOW);
 digitalWriteDirect(LED1, HIGH);
 #ifdef DEBUG
        Serial.print("TC7 timeout");
         //Serial.println(startIndex);
 #endif

```

```
}
```

A.2 RSU

A.2.1 StartServerUDP.java

```
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.concurrent.ConcurrentHashMap;


public class StartServerUDP {
 Scanner scan = new Scanner(System.in);
 Monitor mon = new Monitor();
 long startTime;
 long endTime;
 ConcurrentHashMap<String,RoadConnectionUDP>  clientsList =
     new ConcurrentHashMap<String,RoadConnectionUDP>();
 /**
  * @param args
  */
 public static void main(String[] args) {
  // TODO Auto-generated method stub
  new StartServerUDP().run();
 }
 public void run(){
  Monitor mon = new Monitor();
  CarServerThread carS = new CarServerThread(mon, 5123);
  carS.start();

  int tempInt =0;
  int port=8888;
  try{
   RoadConnectionUDP client1 = new RoadConnectionUDP(
       InetAddress.getByName("192.168.1.166"), port, mon);
   RoadConnectionUDP client2 = new RoadConnectionUDP(
       InetAddress.getByName("192.168.1.167"), port, mon);
   client1.start();
   client2.start();
   clientsList.put("0025806", client1);
   clientsList.put("0025807", client2);

   RSU2Road rsu2road = new RSU2Road(clientsList, mon);
   rsu2road.start();
  }catch (Exception e) {
    // TODO Auto-generated catch block
    System.out.println("fail");
    e.printStackTrace();
  }

```

```
int allTagids[] = {25806,25807,25809};

RoadConnectionUDP picked = null;


while(true){
 if(scan.hasNext()){
  String myLine = scan.next();
  switch (myLine){
  case "list":
   for(String s:clientsList.keySet()){
    System.out.println("tag:"+ s + " : Adress=" +
        clientsList.get(s).toString());

   }
   break;
  case "pick":
   System.out.print("tagNumber:");
   String pick= scan.next();
   picked = clientsList.get(pick);
   if(picked!=null){
    System.out.println("picked index:"+pick + " : "+picked);
   }
   break;
  case "setid":
   if(picked!=null){
    System.out.print("id:");
    picked.setTagId(Integer.parseInt(scan.next()));
   }else{
    System.out.println("no connection picked");
   }
   break;
  case "getid":
   if(picked!=null){
    System.out.println("picked.getTagId()");
   }else{
    System.out.println("no connection picked");
   }
   break;
  case "sendsetid":
   if(picked!=null){
    System.out.print("id:");
    picked.sendSetTagId(Integer.parseInt(scan.next()));
   }else{
    System.out.println("no connection picked");
   }
   break;
  case "sendgetid":
   if(picked!=null){
    picked.sendGetTagId();
```

```
      }else{
       System.out.println("no connection picked");
      }
      break;
     case "prime":
      if(picked!=null){
       System.out.print("time [ms/10 = *100us]:");
       int timeset= Integer.parseInt(scan.next());
       System.out.print("\n");
       if(timeset>0){
        picked.sendPrime(timeset);
       }else{
        picked.sendPrime();
       }
      }else{
       System.out.println("no connection picked");
      }
      break;
     case "stop":
      if(picked!=null){
       picked.sendStop();
       clientsList.remove(picked);
       picked=null;
      }else{
       System.out.println("no connection picked");
      }
      break;
     case "turnon":
      if(picked!=null){
       picked.sendTurnON();
      }else{
       System.out.println("no connection picked");
      }
      break;
     case "turnoff":
      if(picked!=null){
       picked.sendTurnOFF();
      }else{
       System.out.println("no connection picked");
      }
      break;
     case "close":
      mon.removeCar(0);
      break;
     default:
      printHelp();
      break;

    }
   }

```

```
  }
 }
 private void printHelp(){
  System.out.println("----------Help---------");
  System.out.println("h/help   =Print a list of all commands (THIS)");
  System.out.println("list   =Print a list of all connected");
  System.out.println("pick   =pick connection in list to do:");
  System.out.println("  setid id =setid in class");
  System.out.println("  getid   =getid from class");
  System.out.println("  sendsetid id  =send setid to road");
  System.out.println("  sendgetid  =send getid to to road. update class");
  System.out.println("  stop   =disconnect client");
  System.out.println("  turnon   =send getid to to road. update class");
  System.out.println("  turnoff   =send getid to to road. update class");

  System.out.println("-------End Help---------");

 }
 public void printConnected(String s){
  System.out.println(s);
 }

}
```

A.2.2 Monitor.java

```
import java.io.IOException;
import java.util.Iterator;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.ListIterator;
import java.util.ArrayDeque;
import java.util.HashMap;
import java.util.Map;

public class Monitor {

private Map<Integer, Car> carMap;
private ArrayDeque<String[]> latestTagsDetected;
private int curSizeTD = 0;      /* current size of latestTagsDetected */
private int maxCars = 50;
private int nextAvailableID = 0;

public Monitor() {
 carMap = new HashMap<Integer, Car>(10);
```

```
  latestTagsDetected = new ArrayDeque<>();
}

/* called by RecThread when a new car client has connected */
public synchronized int addCar(String id) {
  Car c = new Car(id);
  carMap.put(nextAvailableID, c);
  int softID = nextAvailableID;
  nextAvailableID = nextAvailableID % maxCars;
  notifyAll();
  return softID;
}


public synchronized void removeCar(int id) {
  carMap.get(id).setRemove(true);
  notifyAll();
}

public synchronized boolean shouldContinue(int id) {
  boolean a = carMap.get(id).getRemove();
  if(a) {
    carMap.remove(id);
    notifyAll();
  }
  return !a;
}

/* called by RecThread */
public synchronized void tagDetected(String t1, String t2) {
  String[] ret = {t1,t2};
  latestTagsDetected.add(ret);
  curSizeTD++;
  notifyAll();
}

/* called by RoadSendTestThread */
public synchronized String[] getDetectedTagID() {
  while(curSizeTD == 0) {
    try{
      wait();
    } catch (InterruptedException e) {
      System.out.println("Error: Couldn't wait: " + e.getMessage());
    }
  }
  curSizeTD--;
  return latestTagsDetected.poll();
}

/* Car is an inner class in Monitor */
```

```
 private class Car {

  boolean remove;
  String ID;

  public Car(String id) {
   remove = false;
   ID = id;
  }

  public void setRemove(boolean r) {
   remove = r;
  }

  public boolean getRemove() {
   return remove;
  }

  public String getID() {
   return ID;
  }
 }
}
```

A.2.3 RoadConnectionUDP.java

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.io.UnsupportedEncodingException;
import java.math.BigInteger;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.Queue;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.Lock;


public class RoadConnectionUDP extends Thread {

 private Queue<int[]> sendQue;
 private DatagramSocket socket;
 private InetAddress address;
 private int port;
 private int stage=0;
```

```
private int dataLength=0;
private boolean hexStage =false;
private int[] data = new int[256];
private byte[] dataIn = new byte[256];
private Monitor mon;

private int id=0;
private Object lock = new Object();
private int lastCarEnergyUsed=0;
private boolean newEnergyUsed = false;
final private int PRIME =0X0F;
final private int SETID =0X0A;
final private int GETID =0x0B;
final private int TURNON =0X0C;
final private int TURNOFF =0x0D;
final private int ACKPRIME  =0X06;
final private int ACKSETID  =0X05;
final private int ACKGETID  =0x04;
final private int ACKTURNON  =0X03;
final private int ACKTURNOFF  =0x02;
final private int PING    =0x01;
private boolean stop = false;
private int[] lastMsg;

//Debug
long timeStart =0;
long timeEnd =0;

//constructor
public RoadConnectionUDP(InetAddress address, int port,
    Monitor mon) throws Exception
{
 socket= new DatagramSocket();
 socket.connect(address, port);
 this.address = address;
 this.port = port;
 sendQue = new ConcurrentLinkedQueue<int[]>();
 this.mon=mon;
}
public synchronized void run(){

 while (true) {

  try {

   if(!sendQue.isEmpty()){
    lastMsg=sendQue.poll();
    sendRecive(lastMsg); //protected

   }
   wait();
```

```

	} catch (Exception e) {
	 // TODO Auto-generated catch block
	 e.printStackTrace();
	}
   }

 }
 private void sendRecive(int[] lastMsg2) throws IOException {
  // TODO Auto-generated method stub
  if(lastMsg2==null){

  }
  byte checksumSend;
  DatagramPacket dpSend;
  byte[] dataRec;
  byte [] dataToSend = new byte[((lastMsg2[1])<<1)+6];
  dataToSend[0]='?';
  checksumSend='?';
  int i3=0;
  while (i3 <= (((lastMsg2[1])<<1)+2)){
   System.out.print(i3);

   checksumSend^=(hexEncode((lastMsg2[(i3/2)] & 0xFF) >>4)); // /16
   checksumSend^=(hexEncode(lastMsg2[(i3/2)] & 0xF)); // %16
   dataToSend[i3+1] = (hexEncode((lastMsg2[(i3)/2] & 0xFF) >>4)); // /16
   dataToSend[i3+2] = (hexEncode(lastMsg2[(i3/2)] & 0xF)); // %16
   i3++;
   i3++;
  }
  dataToSend[dataToSend.length-1]=checksumSend;

  boolean retry=true;
  int retrys = 0;
  byte checksum =0;
  dataRec = new byte[255];
  while(retry && retrys < 10){
   retry=false;
   dpSend = new DatagramPacket(dataToSend, dataToSend.length, address,port);

   socket.send(dpSend);
   DatagramPacket dp = new DatagramPacket(dataRec, dataRec.length,address,port);

    try {
     socket.receive(dp);
    } catch (Exception e) {
```

```
    // TODO Auto-generated catch block
    //e.printStackTrace();
    retry=true;
    retrys++;
   }
   if(dp.getLength()!=255){
    checksum =0;
    for(int i=0; i < dp.getLength() ; i++){
          checksum ^= dataRec[i];
    }
    System.out.println("checksum ="+ checksum);
    System.out.println(new String(dataRec));
   }

 }
 if(checksum==0){
      data[0] = hexDecode(dataRec[1]) <<4; // *16
      data[0] += hexDecode(dataRec[2]);
      data[1] = hexDecode(dataRec[3]) <<4; // *16
      data[1] += hexDecode(dataRec[4]);
        for(int i = 0; i < (data[1]<<1);i+=2){
           data[2+(i>>1)] = hexDecode(dataRec[5+i]) <<4; // *16
           data[2+(i>>1)] += hexDecode(dataRec[6+i]);
        }
      instruction(data);
    }
 timeEnd=System.nanoTime();
 System.out.println("  time:" +(timeEnd-timeStart) + "ns\n");
 if(retrys == 10){
  System.out.println("fail, no ack");
 }

}
public String toString(){
 return ""+ address;

}
synchronized public void sendStop() {
 notifyAll();
 stop=true;
 // TODO Auto-generated method stub

}
synchronized public void sendPrime(){
 timeStart = System.nanoTime();
 int[] b= {PRIME,0x01,0x01};
 try {
  sendQue.add(b); //protected
  notifyAll();
 } catch (Exception e) {
```

```
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
 }
 synchronized public void sendPrime(int time100us){
  timeStart = System.nanoTime();

  int[] bytes = intToByteArray(time100us);
  System.out.println("DEBUG primeing, time:" + time100us);
  int[] b= {PRIME,0x02,bytes[2],bytes[3]};
  try {
   sendQue.add(b); //protected
   notifyAll();
  } catch (Exception e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }

 }
 synchronized public void sendTurnON() {
  // TODO Auto-generated method stub
  int[] b= {TURNON,0x01,0x01};
  try {
   sendQue.add(b); //protected
  } catch (Exception e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
  notifyAll();
 }
 synchronized public void sendTurnOFF() {
  // TODO Auto-generated method stub
  int[] b= {TURNOFF,0x01,0x01};
  try {
   sendQue.add(b); //protected
  } catch (Exception e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
  notifyAll();
 }
 synchronized public int getTagId(){
  return id;
 }
 public void sendGetTagId(){
  int[] b= {GETID,0x01,0x01};
  try {
   sendQue.add(b); //protected
  } catch (Exception e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
```

```
    }
  }
  synchronized public void setTagId(int id){
    this.id = id;
  }
  public synchronized void sendSetTagId(int id){
    int[] bytes = intToByteArray(id);
    int[] b =  {SETID,bytes.length};
    int[] sum = new int[b.length + bytes.length];
    System.arraycopy(b, 0, sum, 0, b.length);
    System.arraycopy(bytes, 0, sum, b.length, bytes.length);
    setTagId(id);
    try {
      sendQue.add(sum); //protected
    } catch (Exception e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    notifyAll();
  }
  private int[] intToByteArray(int value) {
    return new int[] {
        (value >>> 24) & 0XFF,
        (value >>> 16) & 0XFF,
        ((value >>> 8) & 0XFF),
        (value & 0xFF)};
  }

  private void instruction(int[] data){

    if(data[0]==ACKGETID){
      int tempid= ((data[2]& 0XFF) << 24) +
          ((data[3]& 0XFF) << 16)  +
          ((data[4] & 0XFF) << 8) +
          ((data[5]) & 0XFF);
      setTagId(tempid);
    }
    if(data[0]==ACKPRIME){

    }

  }

  private String toHex(String arg) {
    try {
      return String.format("%02x", new BigInteger(1, arg.getBytes
          ("UTF-8")));
    } catch (UnsupportedEncodingException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
```

```
   return null;
  }

  byte hexDecode(int c)
  {
   if(c >= '0' && c <= '9')
   {
    return (byte)(c - '0');
   }
   else if(c >= 'a' && c <= 'f')
   {
    return (byte)(c - 'a' + 10);
   }
   else if(c >= 'A' && c <= 'F')
   {
    return (byte)(c - 'A' + 10);
   }
   else
   {
    return 0;
   }
  }
  byte hexEncode(int i, boolean cap)
  {
   if(i >= 0 && i <= 9)
   {
    return (byte) (i + '0');
   }
   else if(i >= 10 && i <= 15)
   {
    if(cap)
    {
     return (byte) (i - 10 + 'A');
    }
    else
    {
     return (byte) (i - 10 + 'a');
    }
   }
   else
   {
    return '0';
   }
  }
  byte hexEncode(int i)
  {
   return hexEncode(i, true);
  }
}
```

A.2.4 CarServerThread.java

```
import java.net.ServerSocket;
import java.net.Socket;
import java.io.IOException;

public class CarServerThread extends Thread {

private Monitor mon;
private int port;

public CarServerThread(Monitor m, int p) {
 mon = m;
 port = p;
}
 public void run(){

  ServerSocket serverSocket = null;
  try {
   serverSocket = new ServerSocket(port);

   while(true){
    new RecThread(serverSocket.accept(), mon).start();
    System.out.println("CarConnected");
    }

   } catch (IOException e) {
    System.err.println("Could not listen on port: " + port);
   System.exit(1);
   }

 }

}
```

A.2.5 RecThread.java

```
import java.net.Socket;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.BufferedWriter;
import java.io.OutputStreamWriter;
import java.io.IOException;
import java.util.Arrays;


public class RecThread extends Thread {

private Monitor mon;
private Socket mySocket = null;
private BufferedReader in;
```

```
private BufferedWriter out;

public RecThread(Socket socket, Monitor m) {
 mySocket = socket;
 mon = m;
 try {
  in = new BufferedReader(new InputStreamReader(mySocket.getInputStream()));
  out = new BufferedWriter(new OutputStreamWriter(mySocket.getOutputStream()));
 } catch (IOException e) {
  System.out.println("Error: Couldn't connect: " + e.getMessage());
 }
}

private int charArrayToInt(char[] data,int start,int end) throws NumberFormatException
{
    int result = 0;
    for (int i = start; i < end; i++)
    {
        int digit = (int)data[i] - (int)'0';
        if ((digit < 0) || (digit > 9)) throw new NumberFormatException();
        result *= 10;
        result += digit;
    }
    return result;
}



 public void run(){

  String speed = "000";
  String tagID, ID;
  int s = 0, t = 0, i = 0;
  int state = 0, softID = 0;
  char ch;
  char[] tagIDArray = new char[7];
  char[] speedArray = new char[3];
  boolean msgNotDone = true;
  boolean alive = true;
  boolean carAdded  = false;

  while(alive){
   try {
    if(in.ready()) {
     msgNotDone = true;
     while(msgNotDone) {
```

```
        ch = (char) in.read();
        //System.out.println("I read char " + ch);
        switch(state) {
         case 0:
          if(ch == '?') {
           state++;
           Arrays.fill(tagIDArray,'0');
           Arrays.fill(speedArray,'0');
          //System.out.println("in case0");
          }
          break;
         case 1:
          //System.out.println("in case1");
          if(ch == '?') {
           state = 0;
          } else if(ch == 'S') {
           state = 2;
          } else if(ch == 'T') {
           state = 3;
          } else if(ch == 'I') {
           state = 4;
          } else if(ch == '!') {
           msgNotDone = false;
           state = 0;
          //System.out.println("I rec !");
          }
          break;
         case 2:
          if(ch == '?') {
           state = 0;
           break;
          }else if(ch == '!') {
           msgNotDone = false;
           state = 0;
           break;
          }

          //System.out.println("in case2");
          speedArray[s] = ch;
          //System.out.println("I put ch " + ch + "in speed at index " + s);
          s++;
          if(s == 3) {
           speed = String.valueOf(speedArray);
           s = 0;
           state = 1;
           System.out.println("speed: " + speed);
          }
          break;
         case 3:
          //System.out.println("in case3");
```

```
            if(ch == '?') {
             state = 0;
             break;
            }else if(ch == '!') {
             msgNotDone = false;
             state = 0;
             break;
            }
            tagIDArray[t] = ch;
            //System.out.println("I put ch " + ch + "in tagID at
                index " + t);
            t++;
            if(t == 7) {
             tagID = String.valueOf(tagIDArray);
             mon.tagDetected(tagID, speed);
             t = 0;
             state = 1;
             System.out.println("tagId: " + tagID);
            }
            break;
           case 4:
            //System.out.println("in case4");
            if(ch == '?') {
             state = 0;
             break;
            }else if(ch == '!') {
             msgNotDone = false;
             state = 0;
             break;
            } else {
             ID = String.valueOf(ch);
             softID = mon.addCar(ID);
             carAdded = true;
             state = 1;
            }
            break;
           }
        }
       }
       if(carAdded && !mon.shouldContinue(softID)) {
        System.out.println("Thread says: I should die");
        try{
         mySocket.close();
         System.out.println("Clientsocket closed");
        } catch (IOException e) {
          e.printStackTrace();
        }
        alive = false;
       }
       Thread.sleep(1);
      } catch (IOException e) {
```

```
      e.printStackTrace();
    } catch (InterruptedException e) {
    }
    }
  }
}
```

A.2.6 RSU2Road.java

```
import java.util.concurrent.ConcurrentHashMap;


public class RSU2Road extends Thread {
  Monitor mon;
  ConcurrentHashMap<String,RoadConnectionUDP>  clientsList;
  public RSU2Road(ConcurrentHashMap<String,RoadConnectionUDP>
      clientsList, Monitor mon){
   this.clientsList = clientsList;
   this.mon = mon;
  }
  public void run(){
   while(true){
    String[] tagInfo = mon.getDetectedTagID();
    if(tagInfo!=null){
     if(tagInfo.length==2){
      RoadConnectionUDP client=clientsList.get(tagInfo[0]);
      if(client!=null){
       int time = (int)(2.6*3.6/ (Integer.parseInt(tagInfo[1]))
           *10000); // time to turn on for 2.6m  (avstand(
           sandare,avtagare) + 1.07m + 1.15m + maginal)
       // 2.6/8 = 0.325m
       if(time<65000){
        System.out.println("time: " + time+ ", tagid: " +
            tagInfo[0]);
        client.sendPrime(time);
       }else{
        System.out.println("To slow speed! time=" + time);
       }
      }
     }
    }
   }
  }
}
```

A.3 Car code

A.3.1 serial.c

```
#include "serialport.h"

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdlib.h>
#include <strings.h>
#include <stdio.h>


/* baudrate settings are defined in <asm/termbits.h>, which is
 * included by <termios.h> */
#ifndef BAUDRATE
#define BAUDRATE B9600
#endif

#define _POSIX_SOURCE 1  /* POSIX compliant source */

static int fd, c, res;
static struct termios oldtio, newtio;
static const char *device;

int serial_init(const char *modemdevice)
{
    /*
     * Open modem device for reading and writing and not as
         controlling tty
     * because we don't want to get killed if linenoise sends
         CTRL-C.
     **/
    device = modemdevice;
    //fd = open (device, O_RDWR | O_NOCTTY | O_NDELAY);
    fd = open (device, O_RDWR | O_NOCTTY );
    if (fd < 0)
      {
   perror (device);
   exit(-1);
      }

    tcgetattr (fd, &oldtio); /* save current serial port
        settings */
    bzero (&newtio, sizeof (newtio)); /* clear struct for new
        port settings */

    /*
```

```
     *BAUDRATE: Set bps rate. You could also use cfsetispeed and cfsetospeed.
     *CRTSCTS : output hardware flow control (only used if the cable has
     *all necessary lines. )
     *CS8      : 8n1 (8bit,no parity,1 stopbit)
     *CLOCAL   : local connection, no modem contol
     *CREAD    : enable receiving characters
     **/
    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;

    /*
     *IGNPAR   : ignore bytes with parity errors
     *ICRNL    : map CR to NL (otherwise a CR input on the other computer
     *           will not terminate input)
     *           otherwise make device raw (no other input processing)
     **/
    newtio.c_iflag = IGNPAR | ICRNL;

    /*
     * Map NL to CR NL in output.
     *                  */
#if 0
    newtio.c_oflag = ONLCR;
#else
    newtio.c_oflag = 0;
#endif


    /*
     * ICANON  : enable canonical input
     *           disable all echo functionality, and don't send signals to calling program
     **/
#if 1
    newtio.c_lflag = ICANON;
#else
    newtio.c_lflag = 0;
#endif

newtio.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);

    /*
     * initialize all control characters
     * default values can be found in /usr/include/termios.h, and are given
     * in the comments, but we don't need them here
     *                                          */
    newtio.c_cc[VINTR] = 0; /* Ctrl-c */
```

```
    newtio.c_cc[VQUIT] = 0; /* Ctrl-\ */
    newtio.c_cc[VERASE] = 0; /* del */
    newtio.c_cc[VKILL] = 0; /* @ */
    newtio.c_cc[VEOF] = 4; /* Ctrl-d */
    newtio.c_cc[VTIME] = 0; /* inter-character timer unused */
    newtio.c_cc[VMIN] = 1; /* blocking read until 1 character
        arrives */
    newtio.c_cc[VSWTC] = 0; /* '\0' */
    newtio.c_cc[VSTART] = 0; /* Ctrl-q */
    newtio.c_cc[VSTOP] = 0; /* Ctrl-s */
    newtio.c_cc[VSUSP] = 0; /* Ctrl-z */
    newtio.c_cc[VEOL] = 0; /* '\0' */
    newtio.c_cc[VREPRINT] = 0; /* Ctrl-r */
    newtio.c_cc[VDISCARD] = 0; /* Ctrl-u */
    newtio.c_cc[VWERASE] = 0; /* Ctrl-w */
    newtio.c_cc[VLNEXT] = 0; /* Ctrl-v */
    newtio.c_cc[VEOL2] = 0; /* '\0' */

    /*
     * now clean the modem line and activate the settings for
         the port
     **/
    tcflush (fd, TCIFLUSH);
    tcsetattr (fd, TCSANOW, &newtio);

    /*
     * terminal settings done, return file descriptor
     **/

    return fd;
}

void serial_cleanup(int ifd){
    if(ifd != fd) {
     fprintf(stderr, "WARNING! file descriptor != the one
         returned by serial_init()\n");
    }
    /* restore the old port settings */
    tcsetattr (ifd, TCSANOW, &oldtio);
}
```

A.3.2 testSync.cc

```
#include "serialport.h"
#include "connection.h"
#include "connectionclosedexception.h"
#include "monitor.h"


#include <iostream>            // std::cout
#include <thread>              // std::thread
```

```
#include <chrono>              // std::chrono::seconds
#include <sys/types.h>
#include <unistd.h>
#include <stdexcept>
#include <vector>

#define BAUDRATE B9600

using namespace std;

/* run method of comTSS thread */
void runTSS(Monitor& mon){

 unsigned char bufIn[100] = {0};
 int bytes_read[100];
 int pos = 0;
 unsigned char STX = 2, ETX = 3;
 vector<unsigned char> send;

 //int fd = serial_init("/dev/ttyACM0");
 int fd = serial_init("/dev/ttyUSB0");

 while(true) {

  //read from RFID antenna
  int i = 0;
  while(true) {
   cout<<"one more time"<<endl;
   i = 0;

   bytes_read[pos] = read(fd, bufIn+pos, 1);

   if(bufIn[pos] == STX) {
    bytes_read[pos] = read(fd, bufIn+pos, 1); //read command
    bytes_read[pos] = read(fd, bufIn+pos, 1);
     cout<<"Tag read: ";
     while(bufIn[pos] != ETX) {
      cout<<bufIn[pos]<<" ";
      send.push_back(bufIn[pos]);
      i++;
      bytes_read[pos] = read(fd, bufIn+pos, 1);
     }
     cout<<endl;
     mon.putMsg(send);
     send.clear();
     bytes_read[pos] = read(fd, bufIn+pos, 1); //read crc
    } this_thread::sleep_for(chrono::milliseconds(5));
  }
 }
 serial_cleanup(fd);
}
```

```

/* run method of comRSURec thread */
void runRSURec(Monitor& mon, shared_ptr<Connection>& conn) {

 unsigned int recv_msg = 0;

 while(true) {
  try{
   recv_msg = conn->read();

  } catch (ConnectionClosedException&) {
    cout << " no reply from server. Exiting." << endl;
    exit(1);
  }
    this_thread::sleep_for(chrono::milliseconds(500));
 }
}

/* run method of comRSUSend thread */
void runcomRSUSend(Monitor& mon, shared_ptr<Connection>& conn,
     unsigned char id){
 vector<unsigned char> curMsg;
 vector<unsigned char> prevMsg;

 try {
  conn->write('?');
  conn->write('I');
  conn->write(id);
  conn->write('!');
 } catch (ConnectionClosedException&) {
  cout << " no reply from server. Exiting." << endl;
   exit(1);
 }

 while(true) {
  mon.getMsg(curMsg);
  if(curMsg != prevMsg){
   cout<<"curMsg != prevMsg"<<endl;

   for(vector<unsigned char>::iterator it = curMsg.begin() ;
       it != curMsg.end(); ++it){
    try{
     conn->write((*it));
     cout<<" "<<*it;

    } catch (ConnectionClosedException&) {
     cout << " no reply from server. Exiting." << endl;
     exit(1);
    }
   }
   cout<<endl;
```

```
    prevMsg = curMsg;
    curMsg.clear();
   } else {
    cout<<"curMsg == prevMsg"<<endl;
   }
    //this_thread::sleep_for(chrono::milliseconds(2000));
  }
}

void runSpeed(Monitor& mon) {
  string line;
  int s = 0;
  while (cin >> s) {
   mon.putSpeed(s);
   cout<<"Speed updated"<<endl;
   this_thread::sleep_for(chrono::milliseconds(500));
  }
}

//int main(){
int main(int argc, char* argv[]) {
  if (argc != 3) {
   cerr << "Usage: myclient host-name port-number" << endl;
   exit(1);
  }

  int port = -1;
  try {
   port = stoi(argv[2]);
  } catch (exception& e) {
   cerr << "Wrong port number. " << e.what() << endl;
   exit(1);
  }

  shared_ptr<Connection> conn(new Connection (argv[1], port));
  if (!conn->isConnected()) {
   cerr << "Connection attempt failed" << endl;
   exit(1);
  }

  Monitor m;

  thread comTSS(runTSS,std::ref(m));
  thread comRSURec(runRSURec,std::ref(m),std::ref(conn));
  thread comRSUSend(runcomRSUSend,std::ref(m),std::ref(conn),'0');
  thread carSpeed(runSpeed,std::ref(m));
  carSpeed.join();
  comTSS.join();
  comRSURec.join();
  comRSUSend.join();
```

```
}
```

A.3.3 monitor.h

```
#ifndef MONITOR_H
#define MONITOR_H

#include <mutex>                // std::mutex, std::unique_lock
#include <condition_variable> // std::condition_variable
#include <vector>
#include <chrono>

class Monitor {

public:
 Monitor();
 void putMsg(std::vector<unsigned char> d);
 void getMsg(std::vector<unsigned char>& d);
 void putSpeed(int v);
 void updatePowerConsumed(unsigned int a);

private:
 std::mutex mtx;
 std::condition_variable cv;
 bool controlCheck = false;        // true means the car is allowed to draw power from the road
 unsigned int powerConsumed = 48;
 int speed=0;
 std::vector<unsigned char> updatedSpeed;
 size_t nextRead = 0, nextWrite = 0, curSize = 0, buffSize = 100;
 std::vector<std::vector<unsigned char>> data;
 std::chrono::high_resolution_clock::time_point t_start, t_end;
 std::clock_t c_start;

};

#endif
```

A.3.4 monitor.cc

```
\end{}
#include "monitor.h"
#include <iostream>             // std::cout
#include <chrono>
#include <ctime>

using namespace std;

```

```
Monitor::Monitor(): data(buffSize){}

/* called by comTSS thread when it has read data from RFID
    antennen */
void Monitor::putMsg(vector<unsigned char> d){
 unique_lock<mutex> lck(mtx);          //take the mutex
 while(curSize == buffSize) {cv.wait(lck);}
 cout<<"in put msg";
 data[nextWrite].push_back('?');
 data[nextWrite].push_back('S');
 char s[4]={'0','0','0','0'};
 sprintf(s,"%03d",speed);
 data[nextWrite].push_back(s[0]);
 data[nextWrite].push_back(s[1]);
 data[nextWrite].push_back(s[2]);
 data[nextWrite].push_back('T');
 data[nextWrite].insert(data[nextWrite].end(), d.begin(), d.
    end());
 data[nextWrite].push_back('!');

 for (vector<unsigned char>::iterator it = (data[nextWrite]).
    begin() ; it != (data[nextWrite]).end(); ++it){
      //cout << " " << hex << static_cast<unsigned int>(*it);
  cout << " " << *it;
 }
 cout<<endl;
 nextWrite = (nextWrite + 1) % buffSize;
 curSize++;
 lck.unlock();                          //release the mutex
 cv.notify_all();
}

/* called by comRSUSend thread */
void Monitor::getMsg(vector<unsigned char>& vec) {

 unique_lock<mutex> lck(mtx);
 while(curSize == 0) {cv.wait(lck);}
 vec = data[nextRead];
 data[nextRead].clear();
 nextRead = (nextRead + 1) % buffSize;
 curSize--;
 //c_start = clock();
    //t_start = chrono::high_resolution_clock::now();
 lck.unlock();
 cv.notify_all();
}

/* called by carSpeed */
void Monitor::putSpeed(int a) {
 unique_lock<mutex> lck(mtx);
 speed = a;
```

```
 char s[4]={'0','0','0','0'};
 sprintf(s,"%03d",speed);
 updatedSpeed.push_back('S');
 updatedSpeed.push_back(s[0]);
 updatedSpeed.push_back(s[1]);
 updatedSpeed.push_back(s[2]);
 lck.unlock();
 cv.notify_all();
}

/* called by comRSURec */
void Monitor::updatePowerConsumed(unsigned int a) {
 unique_lock<mutex> lck(mtx);
 //clock_t c_end = clock();
 //t_end = chrono::high_resolution_clock::now();
 powerConsumed = a;
 cout<<"Power consumed updated: "<< a << endl;
 //cout << chrono::duration<double, milli>(t_end-t_start).count() << " ms\n"<<endl;
 //cout<< 1000.0 * (c_end-c_start) / CLOCKS_PER_SEC << " ms\n"<<endl;
 lck.unlock();
 cv.notify_all();
}
```

A.3.5 connection.h

```
// -----------------------------------------------------------------
//
//        Client/Server communication package
//
//        Connection header file
//
// Change log
// 950323 RH Initial version
// 951212 RH Modified to allow subclassing of class Connection
// 970127 RH Changed "private" to "protected"
// 990125 PH Changed function names: Read -> read, etc.
// 000114 PH int -> bool, virtual destructors, other minor changes
// 010129 PH added void type to initConnection
// 011212 PH changed char* arguments to const char*
//    changed connection closed handling to exception
//    unsigned char instead of char/int in write/read
// 020102 PH split into separate file for each class
// 040421 PH added namespace, new casts, cleaned up a lot
// 050113 PH added deregisterConnection, new registration (vector),
//    added check for server shutdown, many more changes
```

```
//  130515  PH  removed namespace
//
//
    ------------------------------------------------------------------

#ifndef CONNECTION_H
#define CONNECTION_H

class Server;

/* A Connection object represents a connection (a socket)  */
class Connection {
 friend class Server;
public:
 /* Establishes a connection to the computer 'host' via
     the port 'port' */
 Connection(const char* host, int port);

 /* Creates a Connection object which will be initialized
     by the server */
 Connection();

 /* Closes the connection */
 virtual ~Connection();

 /* Returns true if the connection has been established */
 bool isConnected() const;

 /* Writes a character */
 void write(unsigned char ch) const;

 /* Reads a character */
 unsigned char read() const;

protected:
 /* The socket number that this connections communicates on */
 int my_socket;

 /* Set to true when the constructor has called signal()
     to ignore broken pipe. See comment in the constructor */
 static bool ignoresPipeSignals;

 /* Initialization from server, receives socket number s */
 void initConnection(int s);

 /* Server fetches the socket number */
 int getSocket() const;

 /* Prints error message and exits */
 void error(const char* msg) const;
```

```
};

#endif
```

A.3.6 connection.cc

```
// ---------------------------------------------------------------
//
//         Client/Server communication package
//
//          Connection implementation file
//
// Change log
// 950323 RH Initial version
// 951212 RH Modified to allow subclassing of class Connection
// 970127 RH Added extra include to make the file compile under Linux
// 990125 PH Changed function names: Read -> read, ...
// 000114 PH int -> bool, virtual destructors, other minor changes
// 010129 PH added void type to initConnection
// 011212 PH changed char* arguments to const char*
//    changed connection closed handling to exception
//    unsigned char instead of char/int in write/read
// 020102 PH split into separate file for each class
// 040421 PH added namespace, new casts, cleaned up a lot
// 050113 PH added deregisterConnection, new registration (vector),
//    added check for server shutdown, many more changes
// 090127 PH added include of cstdlib, for exit()
//  130515  PH  removed namespace
//
// ---------------------------------------------------------------

#include "connection.h"
#include "connectionclosedexception.h"

#include <iostream>
#include <cstdlib>      /* exit() */
#include <cstring>      /* memcpy() */
#include <csignal>      /* signal() */
#include <sys/types.h>  /* socket(), connect(), read(), write() */
#include <sys/socket.h> /* socket(), connect() */
#include <netdb.h>      /* gethostbyname() */
#include <arpa/inet.h>  /* htons() */
#include <unistd.h>     /* close(), read(), write() */
```

```
#include <sys/uio.h>     /* read(), write() */
#include <netinet/in.h> /* sockaddr_in */

bool Connection::ignoresPipeSignals = false;

Connection::Connection(const char* host, int port) {
 /*
  * Ignore SIGPIPE signals (broken pipe). A broken pipe (only
     ?)
  * occurs in a client, when it tries to write to a dead
     server.
  * When the signal is ignored, ::write() returns -1 as the
     count
  * of written bytes. Connection::write() tests for this and
  * throws ConnectionClosedException when it happens.
  */
 if (!ignoresPipeSignals) {
  signal(SIGPIPE, SIG_IGN);
  ignoresPipeSignals = true;
 }

 my_socket = socket(AF_INET,SOCK_STREAM, 0);
 if (my_socket < 0) {
  my_socket = -1;
  return;
 }

 sockaddr_in server;
 server.sin_family = AF_INET;
 hostent* hp = gethostbyname(host);
 if (hp == 0) {
  my_socket = -1;
  return;
 }

 memcpy(reinterpret_cast<char*>(&server.sin_addr),
     reinterpret_cast<char*>(hp->h_addr),
     hp->h_length);
 server.sin_port = htons(port);
 if (connect(my_socket,
    reinterpret_cast<sockaddr*>(&server),
    sizeof(server)) < 0) {
  my_socket = -1;
 }
}

Connection::Connection() {
 /*
  * See previous constructor for comments.
  */
 if (!ignoresPipeSignals) {
```

```
    signal(SIGPIPE, SIG_IGN);
    ignoresPipeSignals = true;
  }
  my_socket = -1;
}

Connection::~Connection() {
  if (my_socket != -1) {
    close(my_socket);
  }
}

bool Connection::isConnected() const {
  return my_socket != -1;
}

void Connection::write(unsigned char ch) const {
  if (my_socket == -1) {
    error("Write attempted on a not properly opened connection");
  }
  int count = ::write(my_socket, &ch, 1);
  if (count != 1) {
    throw ConnectionClosedException();

  }
}

unsigned char Connection::read() const {
  if (my_socket == -1) {
    error("Read attempted on a not properly opened connection");
  }
  char data = ' ';
  int count = ::read(my_socket, &data, 1);
  if (count != 1) {
    throw ConnectionClosedException();
    return data;
  }
  return data;
}

void Connection::initConnection(int s) {
  my_socket = s;
}

int Connection::getSocket() const {
  return my_socket;
}

void Connection::error(const char* msg) const {
  std::cerr << "Class Connection: " << msg << std::endl;
```

```
  exit(1);
}
```

Bibliography

EPCglobal 2008, EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz. EPCglobal Inc., 2008. Version 1.2.0. Available at http://www.gs1.org/sites/default/files/docs/epc/uhfc1g2_1_2_0-standard-20080511.pdf, [2 December 2015].

Energiläget 2013. Svenska statens energimyndighet, 2013. Available in printed form or at http://www.energimyndigheten.se/statistik/energilaget/.

Fossilfrihet på väg. Statens offentliga utredningar, 2013. ID-nummer: SOU 2013:84.

Traffic Supervisions Systems TSS, 2015. Available at http://www.compprof-rtp.com/tss-tag/, [4 December 2015].

Elways, 2015. Available at http://elways.se, [4 December 2015].

eHighway, Siemens, 2016. Available at http://w3.siemens.se/home/se/sv/Mobility/interurban_mobility/road_solutions/elvagar-klimatsmarta-och-kostnadseffektiva-transporter/Pages/elvagar-klimatsmarta-och-kostnadseffektiva-transporter.aspx, [22 December 2016].

Philip Abrahamsson. *Electro mechanic design and test of conductive electrical road system - overall system design and protection.* Master's thesis, Division of Industrial Electrical Engineering and Automation Faculty of Engineering, Lund University, 2015. CODEN:LUTEDX/(TEIE-5349)/1-53/(2015).

Marcus Andersson. *Electro mechanic design and test of conductive electrical road system - primary power circui.* Master's thesis, Division of Industrial Electrical Engineering and Automation Faculty of Engineering, Lund University, 2014. CODEN:LUTEDX/(TEIE-5348)/1-44(2014).

C.A. Balanis. *Antenna theory.* John Wiley & Sons, Inc., Hoboken, New Jersey, third edition, 2005. ISBN 978-0-471-66782-7.

S. Evdokimov, B. Fabian, O. Günter, Ivantysynova L., and H. Ziekow. *RFID and the Internet of Things: Technology, Applications and Security Challenges,* *Foundation and Trends in Technology, Information and Operations Management.* Publishers Inc., 2010. ISBN 978-1-601-98444-9. Volume 4, no 2.

K. Finkenzeller. *RFID Handbook: Fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication.* John Wiley & Sons, Ltd, third edition, 2010. ISBN 978-0-470-69506-7.

E. H. Hall. *On a new action of the magnet on electric currents.* Technical report, American Journal of Mathematics 2, No. 3, pages 287–292., 1879. Available at http://www.jstor.org/stable/2369245?seq=6#page_scan_tab_contents , [15 January 2016].

Emil Landqvist and Theodor Hallerby. *Thermal Modeling and Power Limitation of the ElOnRoad system - Modelation and simulation of the thermic capabilities.* Master's thesis, Division of Industrial Electrical Engineering and Automation Faculty of Engineering, Lund University, 2015. CODEN:LUTEDX/(TEIE-5350)/1-65/2015).

K&J Magnetics. Surface fields 101, 2016. URL https://www.kjmagnetics.com/blog.asp?p=surface-fields-101. [28 February 2016].

NdFeB Specialists E-magnets UK. *Grades of Neodymium*, 2016. Available at http://www.ndfeb-info.com/neodymium_grades.aspx,[20 February 2016].

Optys Corporation. *OptRFID, High speed reader: reading tags at 43 & 113 km/h.* 2011. Available at https://www.youtube.com/watch?v=n2g_4Do_3R4, [2 December 2015].

Naomi Oreskes. *Vägar och gators utformning, VGU.* Technical report, 2004. Science 03 Dec 2004: Vol. 306, Issue 5702, pp. 1686 DOI: 10.1126/science.1103618.

PTSFS. *Post och telestyrelsens föreskrifter om undantag från tillståndsplikt för användning av vissa radiosändare.* Post och telestyrelse, 2015. Available at http://www.pts.se/upload/Foreskrifter/Radio/PTSFS-2015_4-undantag.pdf.

A.W. Rudge, K. Milne, Olver A.D., and Knight P. *The handbook of antenna design.* Peter Peregrinus Ltd., London, UK, 1982. ISBN 978-0-906-04882-5. Volume 1.

Henrik Fritzon Sund. *Electro mechanic design and test of conductive electrical road system - implementation of the control unit.* Master's thesis, Division of Industrial Electrical Engineering and Automation Faculty of Engineering, Lund University, 2014. CODEN: LUTEDX/(TEIE-5327)/1-121/(2014).

Lee H. Thomas. *The design of CMOS radio-frequency integrated circuits.* Cambridge University Press, second edition, 2004. ISBN 978-0-521-83539-8.

A. Paul Tipler and Mosca Gene. *Physics for scientists and engineers.* W.H. Freeman and Company, New York, sixth edition, 2008.

VGU. *Vägar och gators utformning, VGU.* Technical report, 2004. Available at http://www.trafikverket.se/TrvSeFiler/Foretag/Bygga_och_underhalla/Vag/Vagutformning/Dokument_vag_och_gatuutformning/

Vagar_och_gators_utformning/Sektion_landsbygd-vagrum/sektion_%20landsbygd_vagrum.pdf,[30 November 2015].

R. Wermke, O. Wilmsmeier, and J. Regtmeier. *How fast is fast? RFID identifies objects up to 200 km/h.* Harting Technology Group, 2014a. Available at http://www.harting-rfid.com/fileadmin/harting/documents/rfid/aktuelles/fachartikel/Whitepaper_RFID_highspeed_HARTING.pdf, [2 December 2015].

R. Wermke, O. Wilmsmeier, and J. Regtmeier. *How fast is fast? RFID identifies objects up to 200 km/h.* Harting Technology Group, 2014b. Available at https://www.youtube.com/watch?v=sZ9yEOLw_so, [2 December 2015].

Zhang Xiaoqiang and Tentzeris Manos. *Applications of Fast-Moving RFID Tags in High-speed Railway Systems.* Technical report, 2011. International Journal of Engineering Business Management. Volume 3, No. 1, pp. 27-31.

Printed by Tryckeriet i E-huset, Lund 2017

LUND UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2017-564

http://www.eit.lth.se