IT Technology
Autonomous wireless charging of robot vehicles
Project Report

UNIVERSITY COLLEGE LILLEBAELT

Authors
Dainius Čeliauskas
Stefan Kumaresu

dain0084@edu.eal.dk
stef1571@edu.eal.dk

Monday, June 3, 2019

Abstract

This document describes the development process and results of a project made for presentation in the University College Lillebaelt final exam, in the summer of 2019.

The goal of the project was to design and build an autonomous robot vehicle, capable of charging itself using a system of "smart" wireless charging stations. The stations are aware of their own status - if they are currently being used, they will reject subsequent charging requests from the robot vehicle.

The project is intended to be a demonstration of knowledge gained through the IT Technology Electronics course, as well as a showcase of various theory learned during the education being put to a practical use case.

Table of Contents

Table of Figures

I. Introduction

1. The Issue

Autonomous robots as we know it have existed since 1948 with the creation of the first electronic autonomous robot by William Grey Walter, and have kept improving in both functionality and efficiency ever since. Nowadays, robots can be found in every corner of society, used in almost every industry, public space and household – most of them barely even noticeable by the public. Perhaps the biggest reason most robots have become so ubiquitous and unobtrusive is their capability to work remotely, without being connected to the power grid, using batteries and various wireless charging techniques.

However, when it comes to wireless charging solutions, they are nowhere near as developed compared to the electronics they are supposed to supply power to - traditional wired charging solutions are still the most widespread and efficient ways to deliver power to electronics, including robots. When it comes to robotics and industrial automation, this traditional method is problematic since wired charging requires external human input & labor every time the robot needs to be charged (which, depending on the type of robot, can turn into a daily effort). This major flaw runs counter to the entire philosophy and purpose to automation and robotics solution architecture, and therefore a solution to this flaw is highly sought after.

2. The Approach

In order to design and build a system where an autonomous robot can charge itself, without human input, wireless charging is practically required since it can be used to deliver power based on proximity to the charger. Automating insertion of a cable into a charging port is much more difficult (though not impossible) than having the robot simply get closer to the charging station, and requires a lot more precision and good timing compared to the latter method.

However, even when wireless charging is used, an autonomous robot must be able to actually get close enough to the charging station when needed, and the charging station must be "aware" of its own status: whether it is ready, currently being used, etc. The system, as a result, must have two-way communication between the robot and the "smart" wireless charging station to be able to achieve the desired results. In the end, such an approach should result in a modular, highly scalable & self-sustaining system of autonomous robots and charging stations.

3. Description and purpose

The goal of this project is to develop a wireless charging station with two or more charging pads for autonomous robot vehicles. Currently existing systems for charging robots have not been developed on as much as the robots themselves over time, and using wired ways of charging the robot requires human input, and as a result makes the robots less autonomous. With this system, the robot can go

dain0084@edu.eal.dk
stef1571@edu.eal.dk

to a wireless charger by itself, when it reaches a certain battery level. By doing this the human is no longer needed for the robot to get charged.

The robot in this project should be able to detect and go to a charging station by itself. It should also be able to check whether the charging station is being used already or if it is free to be used.

The purpose of the project is to make a better system for charging robots, than what is available today. Furthermore, the goal of the project is to include all teachings featured throughout the IT Technology course in University College Lillebælt. Lastly, this project is designed to function as a base for any subsequent projects by newer UCL students - since the system has high potential for scalability and new features, the scope of this project can be expanded upon in future generations of academic projects.

4. Aims and objectives

Primary objective: build a system with at least two wireless charging stations and at least one autonomous robot vehicle, where the robot can differentiate between currently used and free charging stations, and be able to get to an unoccupied station by itself.

Secondary objectives:

1: Build and test a proof of concept for the wireless charging solution.

2: Build and test an autonomous robot vehicle powered by a Raspberry Pi, capable of receiving commands from external PC's and other Raspberry Pis for driving and other routines.

3: Design, build and test a "smart" wireless charging station, also capable of sending & receiving commands wirelessly to/from the robot vehicle. In addition, build and test an improved version of the wireless charging solution and use it in the new charging station.

4: Design and test a system where the robot vehicle can identify a free charging station, and drive itself to it for charging.

5. Intended audience

In a broad sense, this document is intended for people with at least basic knowledge in electrical engineering, classical control theory and software development.

In addition, as previously mentioned, this project is designed to be expanded upon in the future, therefore this document is also intended as a helping tool and reference for subsequent projects based on this system.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

6. Scope

This document includes descriptions of the background theory, hardware and software used in the project's two main parts, wireless charging and the robot vehicle. In addition, it provides a full overview of the system, how each part of the project interacts with each other, and the results. Finally, it discusses the development of the project and provides some recommendations regarding any potential future work for it.

7. Responsibilities

Name	**Focus**	**E-mail**
Dainius Čeliauskas	Electronics schematics, testing & coding; wireless charging research and development; networking protocols; PID controller build and tuning	dain0084@edu.eal.dk
Stefan Kumaresu	Electronics builds, testing, measurements and coding; wireless charging research and development; PID controller build and tuning	stef1571@edu.eal.dk

II. Energy Autonomy: Inductive charging

This chapter describes the background theory and practical application of physics and electronics principles that the wireless charging solution is based on. In addition, it discusses ways to improve the solution's efficiency.

1. Background

1) Transformers & Inductive coupling

Electrical circuits are usually thought of as closed loop networks, which give a return path to the current travelling through the circuit. However, if that is the case, how is it possible to transmit power between two electrical circuits without having conductive material or wiring placed between them? The simplest way to find the solution is to investigate how electric transformers work - their sole function is transferring electrical energy between two or more circuits.

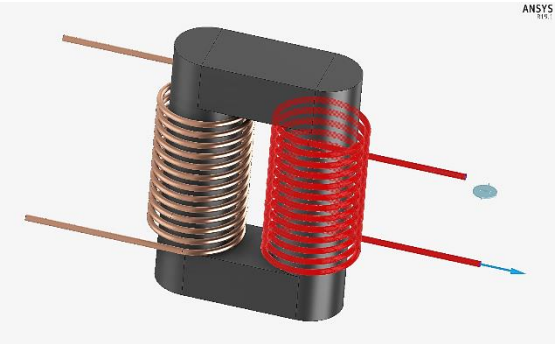


Figure 1: A simple electric transformer model. Source: https://www.ansys.com/blog/electric-transformer-design-simulation-with-ansys-discovery-aim

An electric transformer is an electrical device that uses electromagnetic induction principles to induce electromotive force (EMF) from the coil in one side (known as the primary winding) to the other coil (the secondary winding) wound across the same core. The two windings across each other do not have metallic connections between them, therefore they are electrically isolated from each other.

When electric current changes in one of the transformer coils, it creates a varying magnetic flux, which, in turn, induces a varying EMF across the second transformer coil. The transformer can achieve this by linking two or more circuits together using a common oscillating magnetic circuit, produced by the transformer itself.

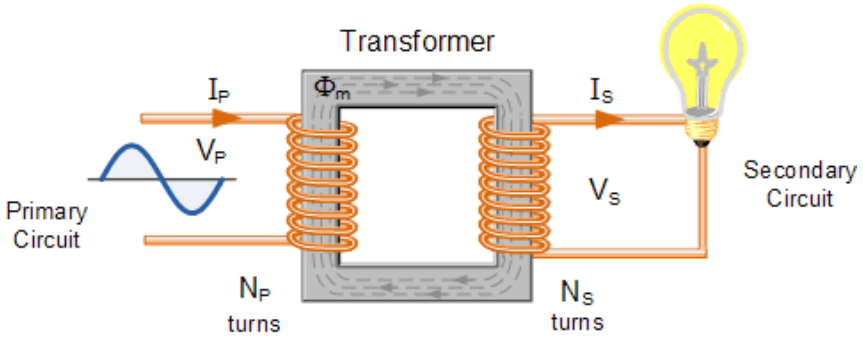


Figure 2: Diagram of a single-phase voltage transformer. Source: https://www.electronics-tutorials.ws/transformer/transformer-basics.html

The primary winding of a transformer is usually connected to an input voltage supply, and transforms the electric current running through the coil into a magnetic field. Meanwhile, the secondary winding converts this magnetic field back into the required output voltage, which is then used to power some form of load.

Measuring and comparing input and output voltages of a transformer is done by looking at the transformer's "turns ratio" - how many coil turns are on the primary winding versus the number of coil turns on the secondary winding. For example, if the primary coil has 15 turns and the secondary coil has 5 turns, the turns ratio will be 15:5 = 3:1. The turns ratio directly influences the input-output voltage ratio - if the input voltage is 9V and the turns ratio is 3:1, the equation for the output voltage will be:

$$V_{out} = \frac{V_{in}}{T.R.}$$
$$\frac{9V}{3} = 3V$$

From the equation it becomes obvious that the turns ratio and voltage ratio are always equal to each other. In addition, the transferred current ratio will always be inversely proportional to the turns and voltage ratios.

Transformers are primarily used to change the output voltage induced in the secondary winding compared to the input voltage applied in the primary winding. When a transformer is used to increase the output voltage, it is known as a step-up transformer, and when it is used to decrease the output voltage, it is called a step-down transformer. In addition, the output voltage can be configured to be the same as the input voltage, and this type of transformer is known as an impedance or isolation transformer. Since transformers are mostly relevant to the project due to their potential to wirelessly transmit electrical power, the main focus will be put on impedance/isolation transformers.

The question remains: if the primary and secondary windings are physically isolated, how does the voltage in the primary winding induce output voltage in the secondary winding in the first place?

When alternating voltage is induced in the primary winding, the current flowing through it created a magnetic field around itself, measured in webers (Φ). This process is called mutual inductance, and it is mathematically defined by Faraday's law of induction.

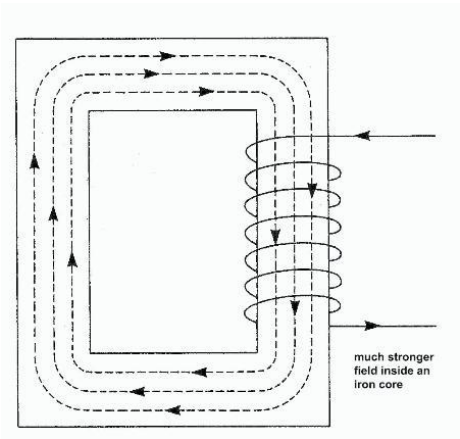


Figure 3: The current flowing through the coil creates magnetic flux (Φ) around the transformer core. Source: http://practicalphysics.org/explaining-how-transformer-works.html

The strength of the magnetic field is proportional to rate of change of the current flow (dΦ/dt). It is this magnetic field that links the two transformer windings, as it changes its strength and direction under the influence of the alternating current. When the "lines" of the magnetic flux circle around the transformer core, they pass through the secondary winding, inducing an EMF defined by the transformer EMF equation of Faraday's Law. Assuming the magnetic field varies sinusoidally ($\Phi = \Phi(max)\ sin(\omega t)$) and the transformer turns ratio is equal to 1, the equations defined the induced voltage in the secondary winding are:

$$E = N\frac{d\Phi}{dt} \textbf{ or } E = N * 2\pi f * \Phi_{max} * cos(2\pi f t)$$

Where E is the induced EMF, N is the number of coil turns in the winding, f is the flux frequency, measured in hertz, and Φ is the amount of magnetic flux, measured in webers. In addition, the induced voltage will have the same frequency as the input voltage from the primary winding.

The resulting output voltage is directly proportional to the number of coil turns in the windings, apart from the peak amplitude of the output voltage decreasing due to core losses and other types of decreases in efficiency (these will be covered in later sections).

Since a transformer requires a varying magnetic flux to function as intended, and the flux is dependent on the rate of change of the input current, as a result it cannot operate on steady state DC voltage - it requires alternating, pulsating or other type of fluctuating voltage or current. DC has no frequency, which results in zero inductive reactance of the primary winding, which in turn results in very low impedance (roughly equal to the resistance of the coil itself). It is entirely possible to cause a short circuit and burn out a steady state DC supply by connecting it to a transformer due to high amounts of current drawn by the winding.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

In conclusion, transformers can be used to change output voltage based on input voltage and some other properties, and deliver electrical power between physically isolated circuits. They are integral to a wide variety of electrical circuits, such as isolated buck-boost converters and blocking oscillators (also known as joule thief circuits). So how can they be applied in wireless charging solutions?

2) Induction chargers & Wireless charging standards

A typical wired charger from a mains wall outlet will usually look like the figure below:

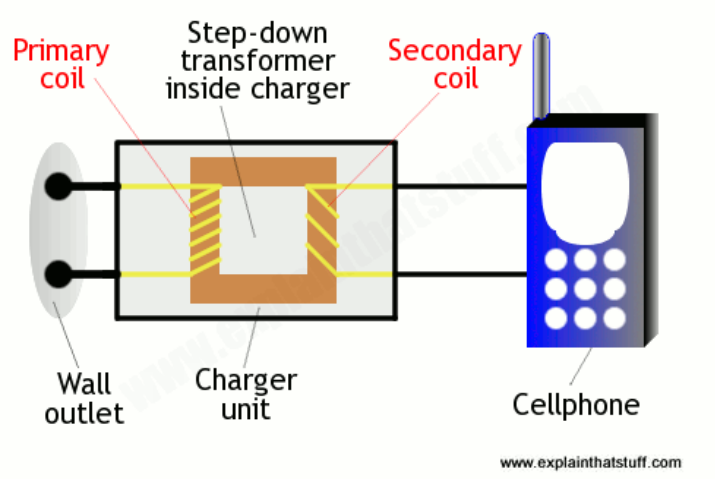


Figure 4: Ordinary electric charging. Source: https://www.explainthatstuff.com/inductionchargers.html

In this setup, the electric transformer is responsible for stepping down the high input voltage coming from the wall outlet (~220-240V AC in Europe) to an output voltage more reasonable for household appliances such as cellphones, vacuum cleaners, etc. In addition, the transformer is entirely contained in the charger unit itself, which is plugged into the appliance.

With a small change, it is possible to convert this setup into a design more resembling current wireless charging solutions:

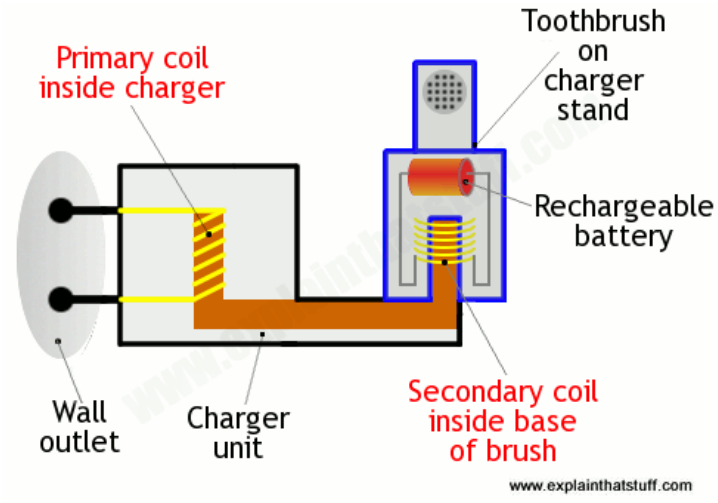


Figure 5: Induction charging design, where the transformer is split between the charger and the electrical appliance (for instance, an electric toothbrush). Source: https://www.explainthatstuff.com/inductionchargers.html

Even though the above figure includes an iron transformer core in its design, it is perfectly possible to have it function without metal cores (such transformers are known as air-core transformers).

This setup, in regards to electromagnetic induction and current flow, is almost completely equivalent to the previous traditional charging example. However, by splitting the transformer and incorporating each half into the charger unit and the appliance itself, the design effectively allows the appliance to be charged wirelessly, using a specialized charging station (referred to as a wireless charging station from now on). This way of charging electrical appliances is called inductive charging.

It is easy to see the advantages of inductive charging compared to traditional direct charging methods - because of no physical connections between the charger unit and the device being charged, corrosion, exposure to the elements and friction can be minimized or removed entirely, which vastly increases durability, portability and safety to users. Of course, it also comes with disadvantages - inductive charging is slower, less efficient and less suited for charging high power electrical systems compared to traditional charging.

Figure 6: A small wireless charging station, charging a modern smartphone. Source: https://faradayscienceshop.com/products/wireless-charging-pad-iphone-android

dain0084@edu.eal.dk
stef1571@edu.eal.dk

Nowadays most small consumer electronic devices (smartphones, toothbrushes, other handheld devices) have some form of support for wireless charging. There are a set of standards defined for wireless consumer electronic charging specifications, and the ones most well adopted are Qi and PMA. For more information on these standards and other existing types of wireless charging, see References.

The most important feature that inductive charging enables for the project is having truly autonomous electric charging systems - wireless charging systems based on this technology can be easily automated and made to function without requiring any human input, provided that the device being charged can navigate itself to the charging station.

2. Application

In the initial development process of the project, a "proof of concept" wireless charging prototype was constructed to test the theory behind inductive charging / wireless power transfer in a practical manner. This chapter discusses the development and results of the experiment, in addition to used hardware, measurements and observations.

The experiment consists of two important parts, the transmitter circuit/coil and the receiver circuit/coil.

1) Transmitter

The parts used for the transmitter coil & circuit were:

- Enameled copper wire
- Breadboard
- 27K ohm resistor
- 2N4401 transistor
- 9V battery
- Assortment of cables

The transmitter coil was made by taking a cylindrical object and making 15 rounds of enameled copper wire around it, one loop and 15 rounds more of enameled copper wire. In total, it is 30 rounds of copper wire with two terminals and one loop terminal. The coil was taped a little so it would not become loose, and the coil was finished by scraping off the insulation on the three terminals.
The transmitter circuit was made on a breadboard, with a resistor, a transistor, a battery and some cables.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

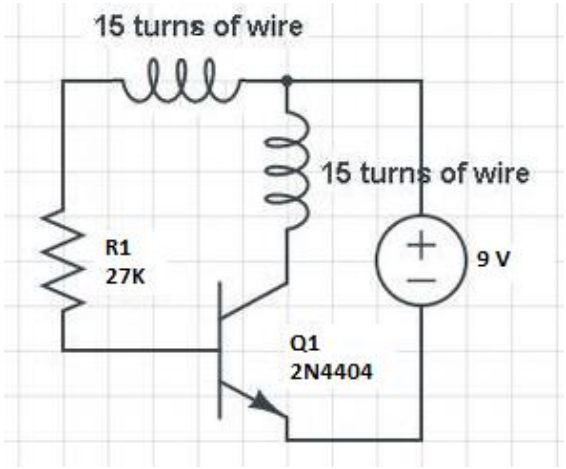


Figure 7: Transmitter circuit diagram. Source: https://www.instructables.com/id/Wireless-electricity-transmission-circuit/

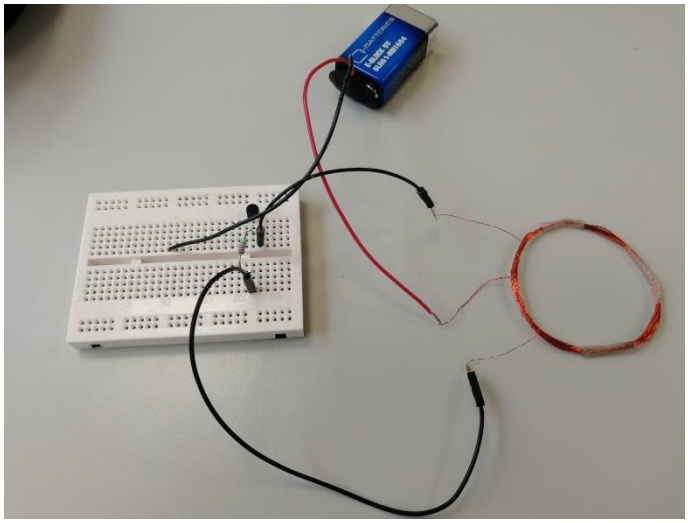


Figure 8: The transmitter coil and circuit on a breadboard.

2) Receiver

The only components needed for making the receiver coil & circuit are enameled copper wire and a light emitting diode. As with the transmitting coil, the enameled copper wire was wound around a round object 30 times, however they can be wound without loop terminals as they are not needed for the receiver circuit. After applying some tape to prevent the coil becoming loose, and scraping off the insulating material on the two terminals along with soldering the light emitting diode onto them, the receiver coil was ready.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

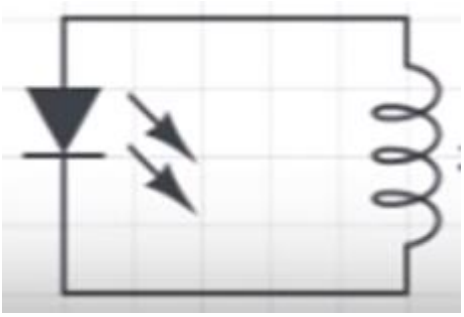


Figure 9: Receiver circuit diagram. Source: https://www.instructables.com/id/Wireless-electricity-transmission-circuit/

Figure 10: Finished receiver coil/circuit.

3) Build and test

The transmitting circuit made for this experiment is an oscillating circuit, similar in design to the "joule thief" circuit - a circuit that will take electricity from a battery and output it at a higher voltage but with lots of intervals in between. As an example, for a light emitting diode that needs 3V to run or light up and a 1.5V battery, the joule thief circuit comes in handy, because it can make the 3V light emitting diode glow with a battery that has a 1.5V EMF. In other words, the joule thief circuit is a step up converter, and as a result the transmitter circuit in the experiment is both an oscillator and a step up converter. And of course, the oscillating/step up converter circuit transfers power to the light emitting diode on the receiver coil wirelessly by using electromagnetic induction, discussed in the previous section.

To sum up the proof of concept experiment, the resulting transformer is air-core, where the transmitter coil receives an oscillating voltage from the oscillator circuit, which induces AC electricity into the second receiver coil and lights up the light emitting diode without having it physically connected to the power supply.

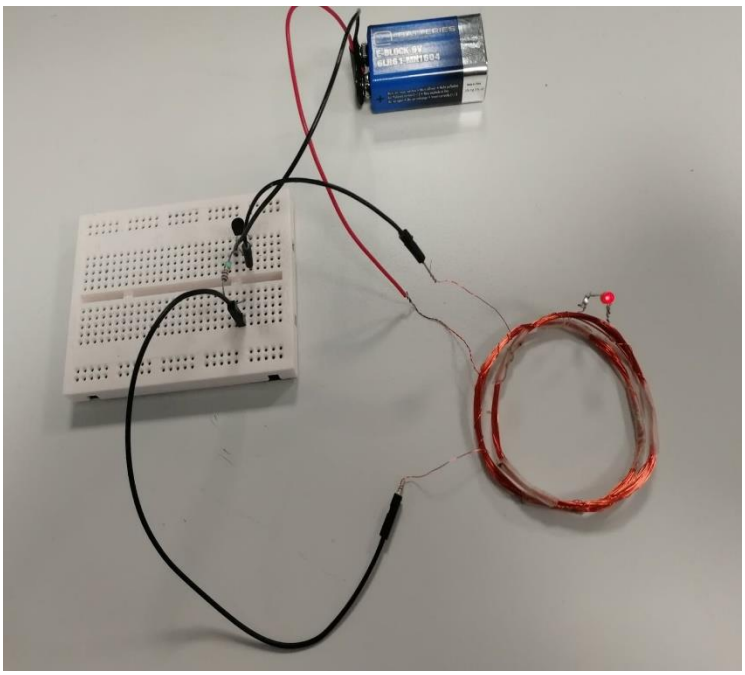


Figure 11: Wireless power transferring proof of concept.

Proof of concept experiment data	Transmitter coil	Receiver coil
Resistance (ohm)	3,6	3,5
Inductance (mH)	0,188	0,177
Diameter (cm)	4	4
Max optimal transmit-receive distance (mm)	10	10

Figure 12: Recorded proof of concept experiment data.

The inductance of the coils was calculated using the equation:

$$L_{coil} = N^2 \mu_0 \mu_r (\frac{D}{2})[ln(\frac{8D}{d}) - 2]$$

Where L_{coil} is the inductance of the coil in henries (H), N is the number of turns, μ_0 is the permeability of free space ($4\pi \times 10-7$), μ_r is the relative permeability of the material, D is the loop material, and d is the wire diameter. See References for the coil inductance calculator which was used for calculations.

3. Optimization

The demonstration shown in the previous section serves as a proof of concept, and confirms that it is indeed possible to create a working wireless charging solution. However, the demonstrated wireless charging setup was only powering a single LED, whereas the goal of the project is to use this kind of design to charge a robot vehicle, a significantly more demanding and complex system.

If this wireless charging solution must be able to power and charge more complex electrical circuits, it is important to maximize its efficiency in delivering electrical power. This section addresses the

dain0084@edu.eal.dk
stef1571@edu.eal.dk

main causes of efficiency loss and ways to reduce or eliminate these causes to get the most out of the existing wireless charging design.

1) Resistance (copper loss) reduction

While electric transformers are generally highly efficient (output voltage is about 97-99% of input voltage), they, like other electronic components, can dissipate power in the form of heat or noise due to electrical resistance. However, transformer coils also increase in resistance if the frequency of the AC signals traveling through it increases. Since power transfer efficiency increases dependent on how high the frequency is, resistance can also increase along with it. The energy that is dissipated by this resistance is known as "copper loss", due to the fact that in making transformer coils, copper wiring is used almost exclusively.

The reason of the increase in resistance of transformer coils when operating at high frequencies is the tendency of the electrons to flow only in the edge of the conducting wire, effectively reducing the conductive diameter of the wire. This tendency is known as the "skin effect" and it is caused by the alternating magnetic flux inducing an electric field in the center of the conductor. The electric field pushes the flowing electrons to the surface of the wire, which decreases the diameter of the actually "used" conductor, which results in increased resistance.

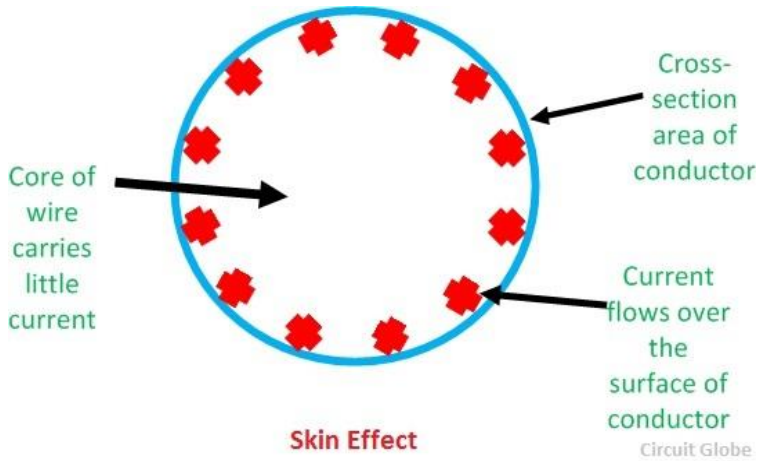


Figure 13: Visual representation of the skin effect. Source: https://circuitglobe.com/skin-effect.html

To minimize the skin effect without having to reduce frequency and therefore power transfer, the most common methods are hollowing out the wire, resulting in increased surface area of the current, or using Litz wire, a specialized multi-strand wire consisting of several thin woven strands that are electrically isolated from each other.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

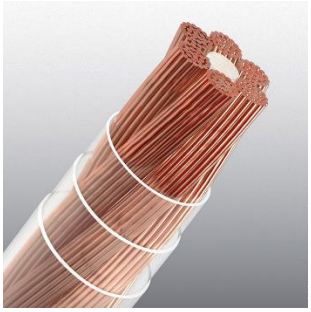


Figure 14: Litz wire cross-section. Source: https://www.deeterelectronics.com/product/litz-wire/litz-wire/

2) Coupling coefficient increase

Another important factor to consider when optimizing inductive charging is the coupling coefficient, denoted by k. It is equal to the fraction of the magnetic flux produced by the primary transformer winding that passes through the secondary winding when it is open circuited.
When all the flux produced by the primary winding passes through the secondary receiver winding (i.e. no leakage flux), k = 1.

When the flux does not align at all between the windings, they are magnetically isolated and k = 0.

The equation for the coupling coefficient is:

$$k = M/\sqrt{L_1 L_2}$$

Where M is the mutual inductance between the coils, L1 is the inductance of the first coil and L2 is the inductance of the second coil.

As a general rule, the mutual inductance changes based on the geometry and the range between the coils - for maximum coupling, they are wide and flat shaped, and the closer they are to each other, the higher their efficiency will be as a result. In addition, if there is enough available space and acceptable weight for a wireless charging solution, ferrite cores can greatly improve mutual inductance, however they are not always practical for small wireless devices. For more information about mutual inductance and related properties, see References.

The actual efficiency and power transfer of the transformer is roughly equal to k^2, therefore the higher the coefficient, the more efficient the transformer will be as a result.

These are the main methods for improving efficiency for ordinary inductive charging systems. However, modern solutions also increase efficiency and effective range by augmenting the design with use of resonant inductive coupling.

3) Resonant inductive coupling

The previously discussed wireless charging design works on the principle of a primary transformer winding generating a magnetic field, and a secondary "collector" winding crossing over as much of the magnetic flux as possible, so that the output voltage of the circuit is as close as possible as that of the input voltage. This setup results in the secondary winding having to be extremely close to the primary winding, and usually requires use of a magnetic core. Such a design becomes highly inefficient over larger distances and wastes a lot of energy.

A significant improvement to efficiency of the design can be achieved by utilizing resonance. Capacitors can be mounted in parallel to each transformer coil to form resonant (LC) circuits, and if the transmitting coil's output frequency matches that of the receiving coil's resonant frequency, the effective range of the wireless charging solution can be increased drastically (more than a couple of meters), without losing a significant amount of efficiency.

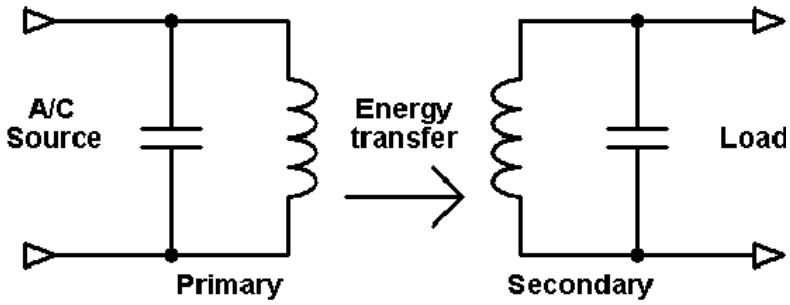


Figure 15: Resonant inductive coupling circuit design, with capacitors in parallel to both coils. Source: https://www.semanticscholar.org/paper/Maximizing-Efficiency-of-Wireless-Power-Transfer-Liu/5038d259157c9fe205259f3705ac856fd1e7e021/figure/0

Each resonant circuit is tuned to resonate at the same resonant frequency, which results in vastly increased coupling and power transfer over greater distances. Interestingly, the actual efficiency of these circuits is slightly lower than that of non-resonant inductive coupling designs, however as distance between the coils increases, it decreases at a much slower rate in comparison.

The effect the capacitors have on the inductively coupled circuits is increase of their Q factor, which is a parameter comparing a system's oscillation frequency to the system's energy dissipation rate. In general terms, the higher a system's Q factor is, the lower the frequency bandwidth becomes, while the resonating signal decays much slower as well. Resonators with a higher Q factor transfer energy much faster than they lose energy due to resistance or other causes. In addition, due to the narrower bandwidth of the resonant circuits, they are coupled much more intensely than non-resonant counterparts and are more immune to noise, while reducing power losses from nearby conductors absorbing the magnetic field. For more information on the Q factor and its applications, see References.

By utilizing the design improvements detailed above, the wireless charging solution can be made vastly more efficient and work at far greater ranges. It can be concluded that the proof of concept

application shown in the previous section still has room for improvement, and has immense potential for not just the project, but also solving many challenges currently existing worldwide.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

III. Autonomous Foraging: Robot vehicle and Charging station

This chapter describes the second part of the project - building and testing the autonomous robot vehicle and smart station system, as well a more in-depth look at the scripts and hardware used.

1. Components

This section discusses the main hardware and software components used in the project, and how they were used.

Minor hardware components used in the project such as capacitors, resistors, cables, etc. are not discussed explicitly.

1) Raspberry Pi 3 Model B

The Raspberry Pi is a multi-purpose single-board computer originally designed as an educational tool for computer science projects. The version used for the project is the Raspberry Pi 3 Model B, released in 2016 by the Raspberry Pi Foundation.

Figure 16: Raspberry Pi 3 Model B. Source: https://www.pcmag.com/review/342705/raspberry-pi-3-model-b

This particular model has quite a few useful features that were utilized in some way. It has an integrated Wi-Fi module and antenna, a micro USB port for 5V input power, and a 40-pin GPIO set for controlling various peripherals.

In the robot vehicle, the Raspberry Pi is responsible for reading sensor input, controlling the motor driver, communicating with the charging stations and maneuvering the vehicle to the stations when charging is required.

In the charging stations, the Raspberry Pi is used primarily to enable communications with the robot vehicle (therefore making it “smart”), as well as enabling/disabling the connected wireless transmitter coil.

The Raspberry Pis used for the project use the Raspbian Stretch Lite OS since no parts of the project use a display, therefore the GUI from default Raspbian Stretch cannot be used. Since there

are no traditional input devices connected to any Raspberry Pi either, making it “headless”, interfacing with the Raspberry will be done by remote connection using SSH.

In addition, the RPi.GPIO Python package is used to control the Raspberry GPIO through Python scripts, such as changing pulse-width modulation (PWM) of the motor enable pins, detecting sensor input and powering various connected components. It normally comes with Raspbian by default, however this version only works with Python 2.7. The scripts for the robot vehicle and charging station (discussed in next section) are designed to be run with Python 3.5.3 and above, therefore to download the correct RPi.GPIO version it must be download and installed manually:

```
sudo apt-get install python3-rpi.gpio
```

It should be noted that the Raspberry Pi was chosen as the main control unit for all systems of the project due to significant time and budget constraints. The Pi could be set up and bring the project to a working state quickly, but for better results a more suitable control board could be used. For recommendations, see “Discussion and recommendations”.

2) 4WD robot chassis with motors

Figure 17: The robot chassis set. Source: https://www.dhgate.com/product/lcll-new-4wd-robot-smart-car-chassis-kits/420610010.html

This type of chassis is fairly common in projects of this kind. It includes four ~60mm diameter wheels, and four DC 120:1 gear motors.
The motor rating for the motors at 5V is 1kg*cm torque and 83 RPM.

3) Motor driver

The motor driver circuit is used in the robot vehicle to control the 4 onboard DC motors.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

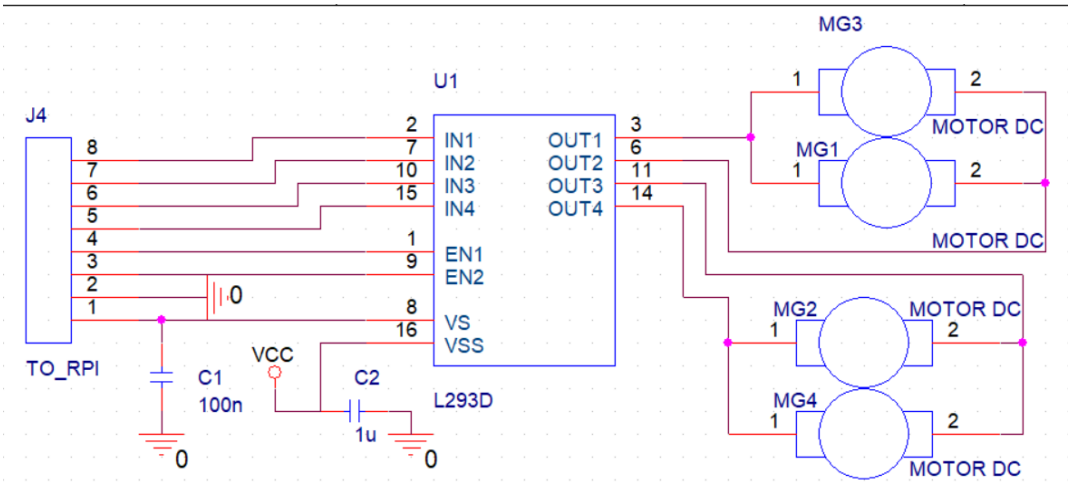


Figure 18: Robot vehicle motor driver circuit schematic.

The main part of the circuit is the L293DNE integrated circuit. The L293DNE is a quadruple half-H driver chip, and in this circuit it is used for bidirectional control of two pairs of parallel DC motors. It supports separate power supplies for its logic (VCC1) and for powering the motors themselves (VCC2). VCC1 can be up to ~7V input, and VCC2 can be provided up to 36V input, while giving up to 1A of bidirectional current for driving the motors. If the motors and the power supply needs to be scaled up to above 36V input/output, it will require usage of a different motor driver, such as the L298 dual full-bridge driver, or a custom built H-bridge solution. The L293DNE was chosen because of its simplicity, easy accessibility and implementation.

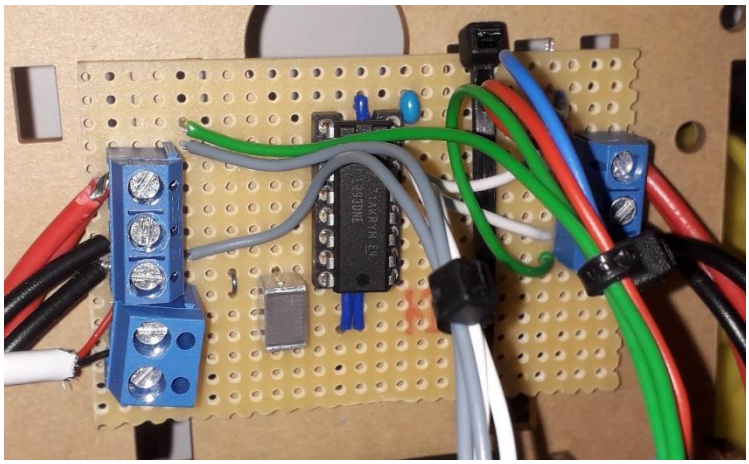


Figure 19: The motor driver circuit on the robot vehicle.

The L293DNE connects to the Raspberry Pi GPIO with 3 pins for each pair of motors: two control pins for powering and changing the direction of the motor spin, and one pin for enabling/disabling the control pins, which is used in changing the motors' duty cycles using PWM. This enables the Pi's GPIO to change the motors' state, direction and speed through scripting, effectively driving the vehicle.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

The datasheet of the L293 IC recommends using bypass capacitors of 0.1 uF or greater on VCC1 and VCC2 pins (see References), presumably to filter out AC noise, therefore they are also incorporated into the circuit design.

4) Line sensor module

The line sensor module circuit design shown below is used to check the reflectivity of the surface under the robot vehicle, and use the retrieved information to navigate the vehicle through a pre-made path (a black colored line on a white colored mat). The robot car uses 3 of these sensor modules.

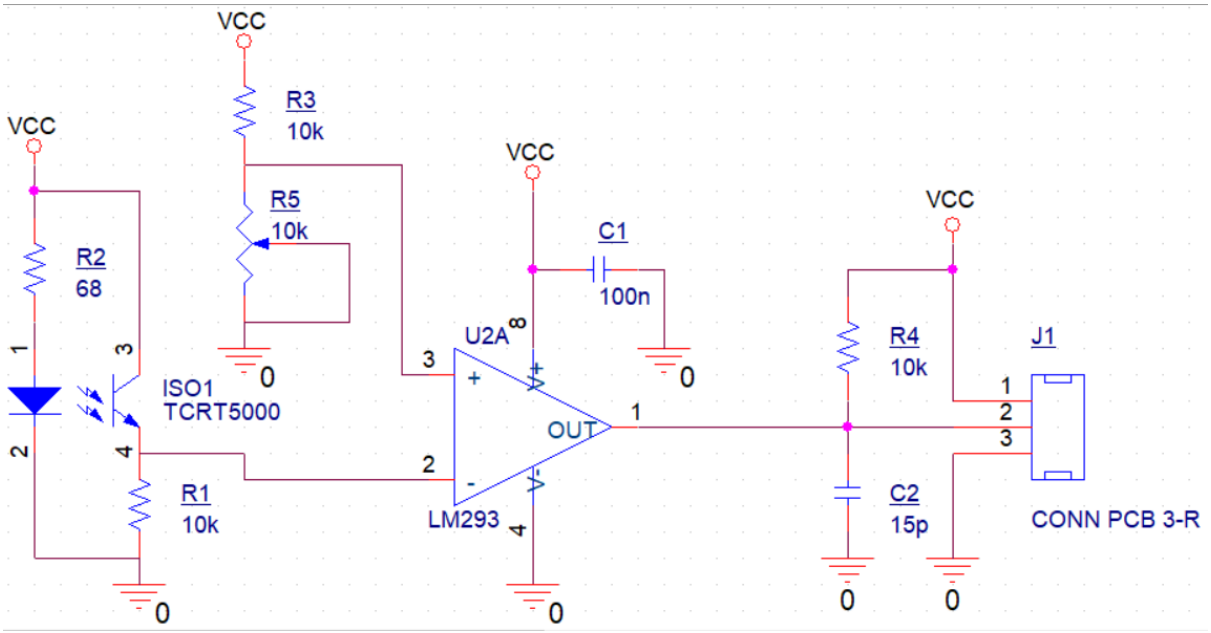


Figure 20: The line sensor module circuit schematic.

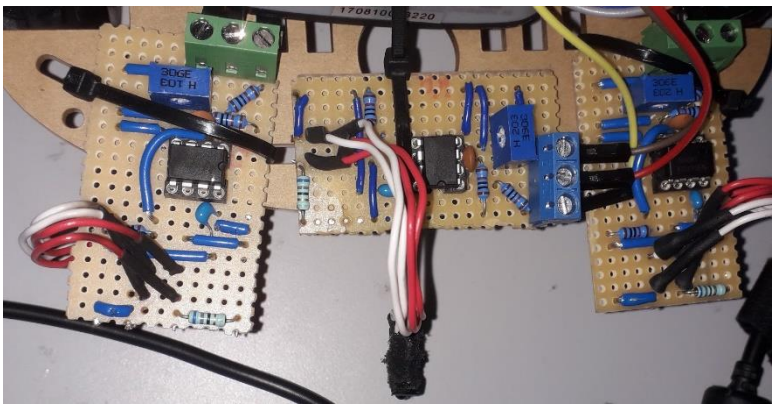


Figure 21: The 3 line sensor modules on the robot vehicle.

The two most important parts of the circuit are the TCRT5000 reflective sensors and the LM293 comparators. The TCRT5000 is a package consisting of an IR diode and a phototransistor, which passes or blocks electric current based on how much IR light from the diode is reflected back at the diode. The output of the sensor is analog and not suited for reading with the Raspberry GPIO

dain0084@edu.eal.dk
stef1571@edu.eal.dk

directly, which is why a comparator such as the LM293 is necessary. The LM293 compares the output voltage from the sensor with a reference voltage (in this case, 0V) and outputs a HIGH or LOW digital signal based on sensor readings. In practice, the sensor module sends a HIGH signal when the sensor is pointed at a black colored surface, and a LOW signal when it is facing a white colored surface. This signal is much more suited for reading with the Raspberry GPIO, and can be used in scripting to evaluate and perform various decisions.

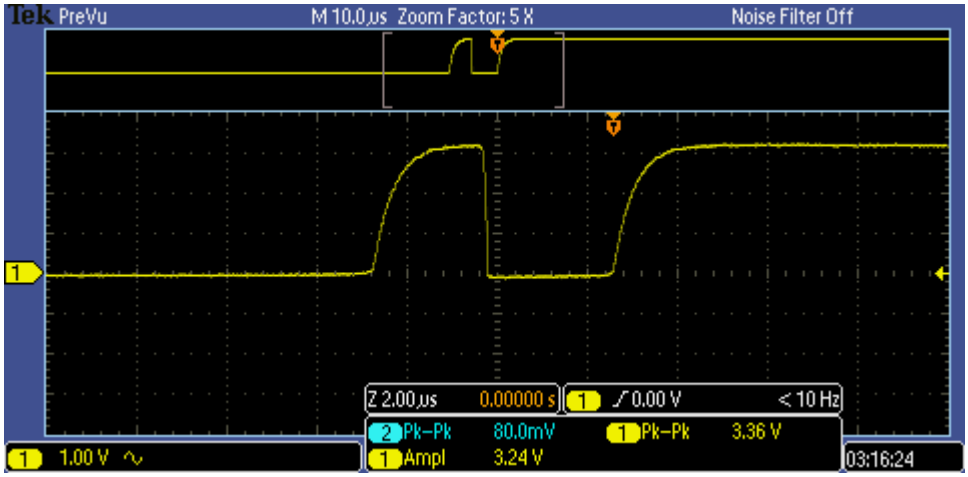


Figure 22: Testing one of the sensor modules with an oscilloscope, going from white to black surface.

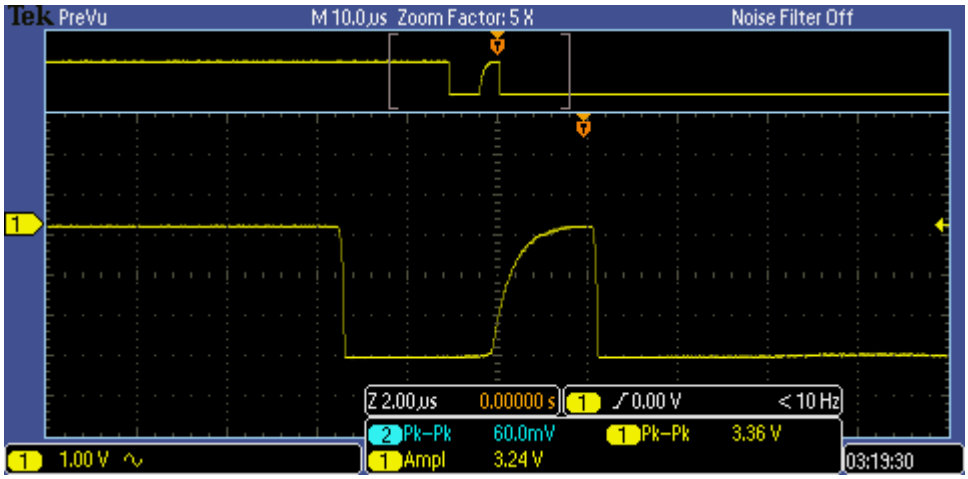


Figure 23: Testing one of the sensor modules with an oscilloscope, going from black to white surface.

5) Trust Urban Omni Powerbank

This brand of powerbank was used in the project to power the robot vehicle without connecting it to a wall outlet or using other wired methods of powering it.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

Figure 24: The Trust Urban Omni powerbank. Source: https://www.trust.com/en/product/21858-omni-ultra-fast-powerbank-10-000-mah-usb-c

The powerbank has a 10000 mAh battery with a level indicator, one micro-USB input port for charging, one USB output port and one input/output USB-C port. The battery has enough capacity to supply power to the robot vehicle system for ~5 hours.

In the robot vehicle, the USB output is connected to the motor driver and supplies 5V to the motors, and the USB-C port is powering the Raspberry Pi with 5V. The micro-USB input port is connected to the wireless receiver coil, which charges the powerbank through the port when current is induced in it by the station's transmitter coil. The maximum current the micro-USB input can take is 2.4 A, which is well within the project's limits as the receiver coil supplies up to 1 A of current.

After some testing, it was found out that when the powerbank is being charged, its output voltage becomes substantially reduced, which proved to be a problem for the project. For more information, see the "Problems and solutions" section in the "Results" chapter.

6) Wireless Charging Module 5V/1A

This type of wireless charging module set was utilized in the robot vehicle and the charging stations. They are sold by DFRobot & Seeed Studios (for links to their main websites, see References). The package contains two parts, a transmitter coil with a circuit and a receiver coil with a circuit.

These modules use inductive charging to transfer power wirelessly between them (for more information on how they work, see the "Energy Autonomy: Inductive charging" chapter). The wireless charging modules can deliver a typical power output of 5V at 1 A, or a max power output of 5V at 1.2 A.

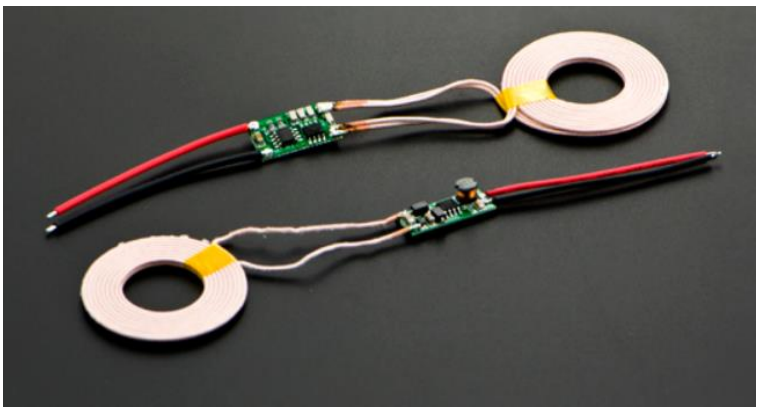


Figure 25: The transmitting and receiving charging coils that were used. Source: https://www.dfrobot.com/product-1284.html

Let's take a look at the transmitter coil and circuit:

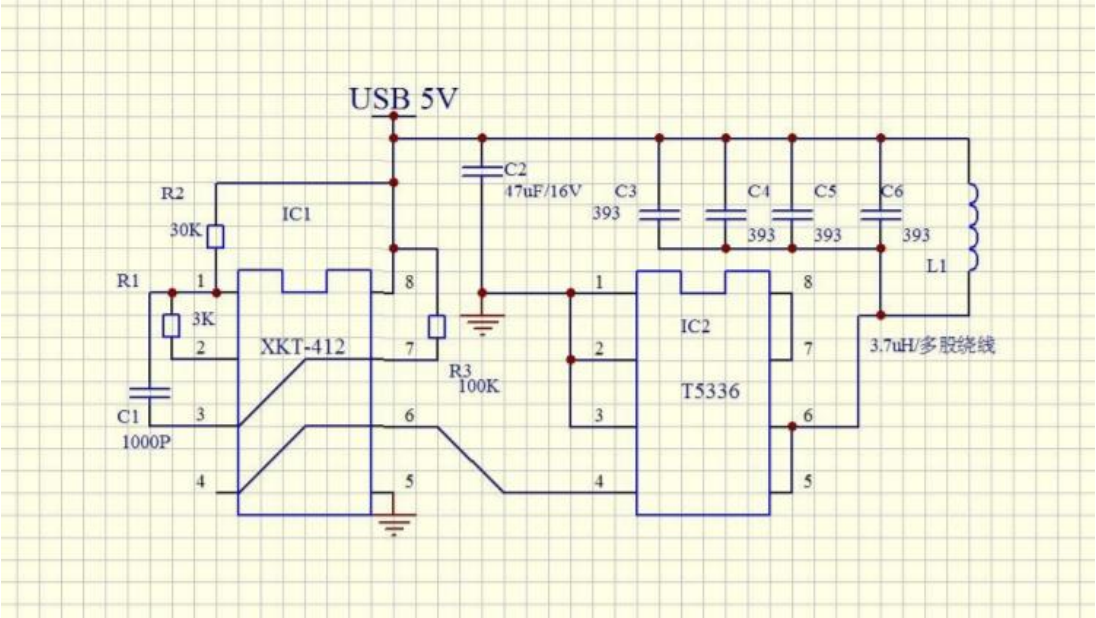


Figure 26: The schematic of the transmitter coil. Source: https://github.com/Arduinolibrary/DFRobot_wireless_charging/raw/master/DFR0362_Schematic.pdf

The first noticeable components of the transmitter circuit are the two integrated circuits (IC). The first IC is the XKT-412, which is a wireless power transfer IC used in wireless charging modules, and the T5336, another integrated chip used for wireless power transferring. The XKT-412 chip can provide current in the amount of 600 mA and the T5336 can provide the last 300-400 mA of current, therefore the total output is about 1 A.

The documentation for these chip is sparse in English because they are produced in China and the producer of these ICs does not provide datasheets in English, only in Chinese. As a result it is difficult to determine some of the functionality of the circuits.

Let’s examine the receiver coil circuit:

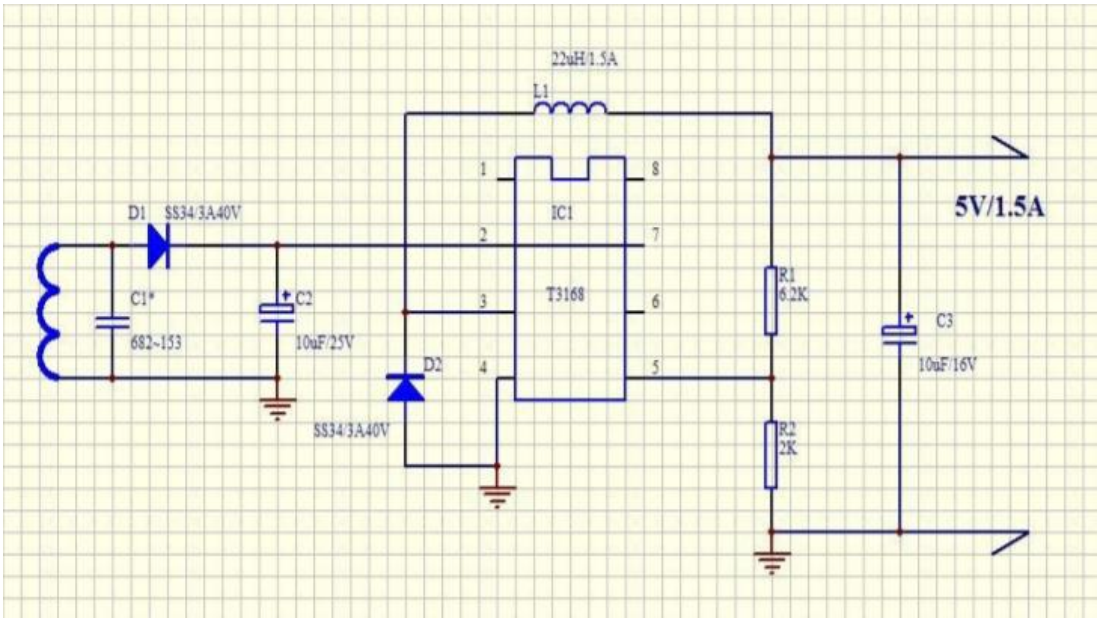


Figure 27: The schematic of the receiver coil. Source: https://github.com/Arduinolibrary/DFRobot_wireless_charging/raw/master/DFR0362_Schematic.pdf

The first thing that is noticeable on the receiver circuit, is again the integrated circuit in the center of the schematic. This is a T3168 chip. It is a wireless charging receiving IC and again the documentation for this chip is sparse, since it is only available in Chinese.

This set of coils can supply the Trust powerbank input with enough power to enable the battery level indicator and start the charging process of the robot car. However, in order to solve the issue with the powerbank (for more information, refer to "Problems and solutions") a solution for toggling the coil on and off with the Raspberry was required. This was achieved by using MOSFET switches on the charging stations:

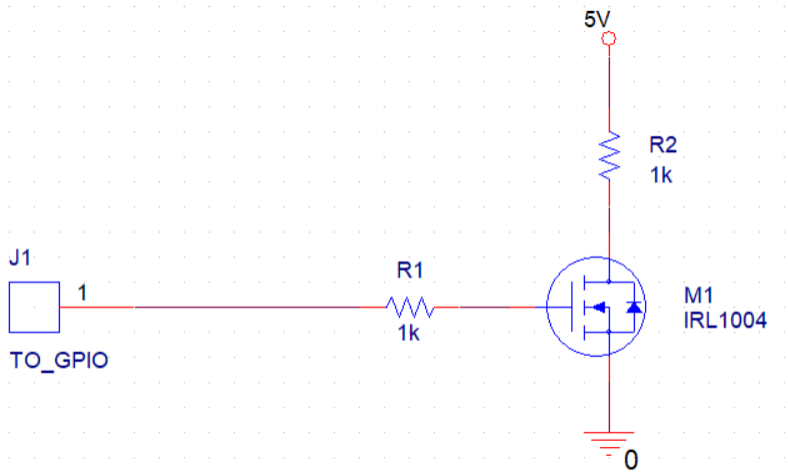


Figure 28: The MOSFET switch designed to enable toggling the transmitter coil on/off with the Raspberry GPIO.

The MOSFET switch (in this circuit, the IRL1004 power MOSFET was used) is a circuit utilized in the charging stations that controls the flow of current to the transmitter coils. This is achieved by triggering the MOSFET gates with the GPIOs of the Raspberry Pi. When the Raspberry GPIO is HIGH or 3.3V, the MOSFET is closed, but when the GPIO goes LOW or 0V, the MOSFET gate is opened and sends approx. 5V & 1A through itself to the transmitter coils. This is necessary since the Raspberry GPIO cannot sink or source more than 16 mA (for more about the Raspberry GPIO electrical specifications, see References), which is not nearly enough for the coil to operate as intended. As a result, the Raspberry GPIO is able to control vast amounts of current flow with its GPIO output signal.

7) Force Sensitive Resistor (pressure pad)

This type of force sensitive resistor (FSR) was used in the charging station design to determine if the robot vehicle is present and parked in the correct spot:

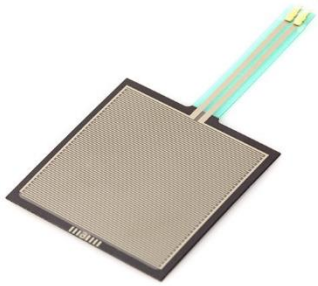


Figure 29: The FSR used in the project. Source: https://let-elektronik.dk/shop/1480-kraft/9376-force-sensitive-resistor---square/?fbclid=IwAR0XM4ACCSTodcAvio1r0SUP8nDY4QVbHNRoVUHvPXkcxyWDUAdyTt-aY4A

The FSR varies its resistance depending on how much pressure is applied to the flat square surface - resistance is high when there is low pressure, and low when some pressure is put on it. This resistance can be measured by the Raspberry GPIO, and the resulting value used in scripting to make certain decisions.

The FSR is placed in such a way that when the robot vehicle approaches the charging station and transmitter coil, it becomes triggered by the wheels of the vehicle applying pressure to it. The resistance drop in the FSR is measured by the GPIO and the script in the Raspberry Pi, which as a result sends a command to the vehicle to stop moving, and toggles the transmitter coil on. Therefore, the FSR allows the charging station to recognize whether or not it is being used, and control the robot vehicle appropriately.

2. Software

1) Setting up the Raspberry Pis

Since none of the Raspberry Pis used in the project have connected displays, they must be connected to remotely using SSH (secure shell) connections. In order to successfully connect to a

Raspberry Pi wirelessly using an external device, both the Pi and the device must be on the same network. However, because most networks nowadays are password or PEAP protected, the Pi is unable to connect to them unless the correct SSID and password are passed to the Pi in some way.

If using Raspbian, the information needed for the Pi to successfully connect to networks is in the `wpa_supplicant.conf` file located in the `/etc/network/interfaces/` or `/etc/wpa_supplicant/` directory. The file's contents should look like the example below:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=GB

network={
      ssid="[Wifi network name]"
      psk="[Wifi network password]"
}
```

If `ssid` and `psk` is substituted with a local network's name and password, the Raspberry will automatically connect to this network when booting up.

The Pi can also attempt to connect to multiple networks if the previous connection was not successful, using the `priority` keyword:

```
network={
      ssid="[Wi-Fi network name]"
      psk="[Wi-Fi network password]"
      priority=1
}
network={
      ssid="[Wi-Fi network name]"
      psk="[Wi-Fi network password]"
      priority=2
}
...
```

If the network is PEAP protected (for example, a school Wi-Fi network requiring a valid username and password), the login information can be entered like this:

```
network={
      ssid="[Wi-Fi network name]"
      proto=RSN
      key_mgmt=WPA-EAP
      pairwise=CCMP
      auth_alg=OPEN
      eap=PEAP
      identity="[Wi-Fi unique identity]"
```

```
    password="[MD4 hash string of Wi-Fi unique password]"
    phase1="peaplabel=0"
    phase2="auth=MSCHAPV2"
}
```

The MD4 hash string can be retrieved using an online MD4 hash generator, or by running this command in the Raspberry and viewing the `hash.txt` output:

```
echo -n 'YOUR_REAL_PASSWORD' | iconv -t utf16le | openssl md4 > hash.txt
```

This method may not always work correctly, however. If the Raspberry is still unable to connect to a PEAP protected network after using this method, another option is to log in the network with a mobile phone, then use it as a mobile hotspot, and specify the hotspot login information in the `wpa_supplicant` file instead.

The next step is actually copying this file to the Raspberry so that it can connect to the specified network(s). The file can be edited using the Raspberry terminal itself, but that first requires having access to the terminal, either by using a display, keyboard and mouse, or by remotely connecting to the Raspberry through SSH, which in itself requires the Raspberry to be connected to a network. Therefore, this step could cause some issues if unprepared. If no display, keyboard and mouse are available, there are two main methods available for uploading the file to the Raspberry.

The first method requires an Ethernet cable which is connected between a PC and the Raspberry. When the PC is connected to the network, the cable bridges the connection between the two devices, which enables SSH connection and access to the Raspberry terminal. For more detailed information on this method, see References.

The second method requires a PC with an in-built microSD port or an USB to microSD adapter. When Raspbian is successfully installed on the microSD card and inserted to the PC by using the port or an adapter, the `wpa_supplicant` file can be copied to the `boot` partition of the card. When Raspbian is booted on the Raspberry again, this file will be automatically moved to the correct directory and used to connect to the specified network.

Note that SSH is disabled in Raspbian by default. It can be enabled from the Raspberry terminal or by copying a blank “ssh” file to the `boot` partition of the microSD card.

Once the external device and the Raspberry are connected and Raspbian is updated, the setup is complete, and the Raspberry is ready for running the project’s software.

2) Robot vehicle code

The flowchart below demonstrates the main algorithm of the robot vehicle:

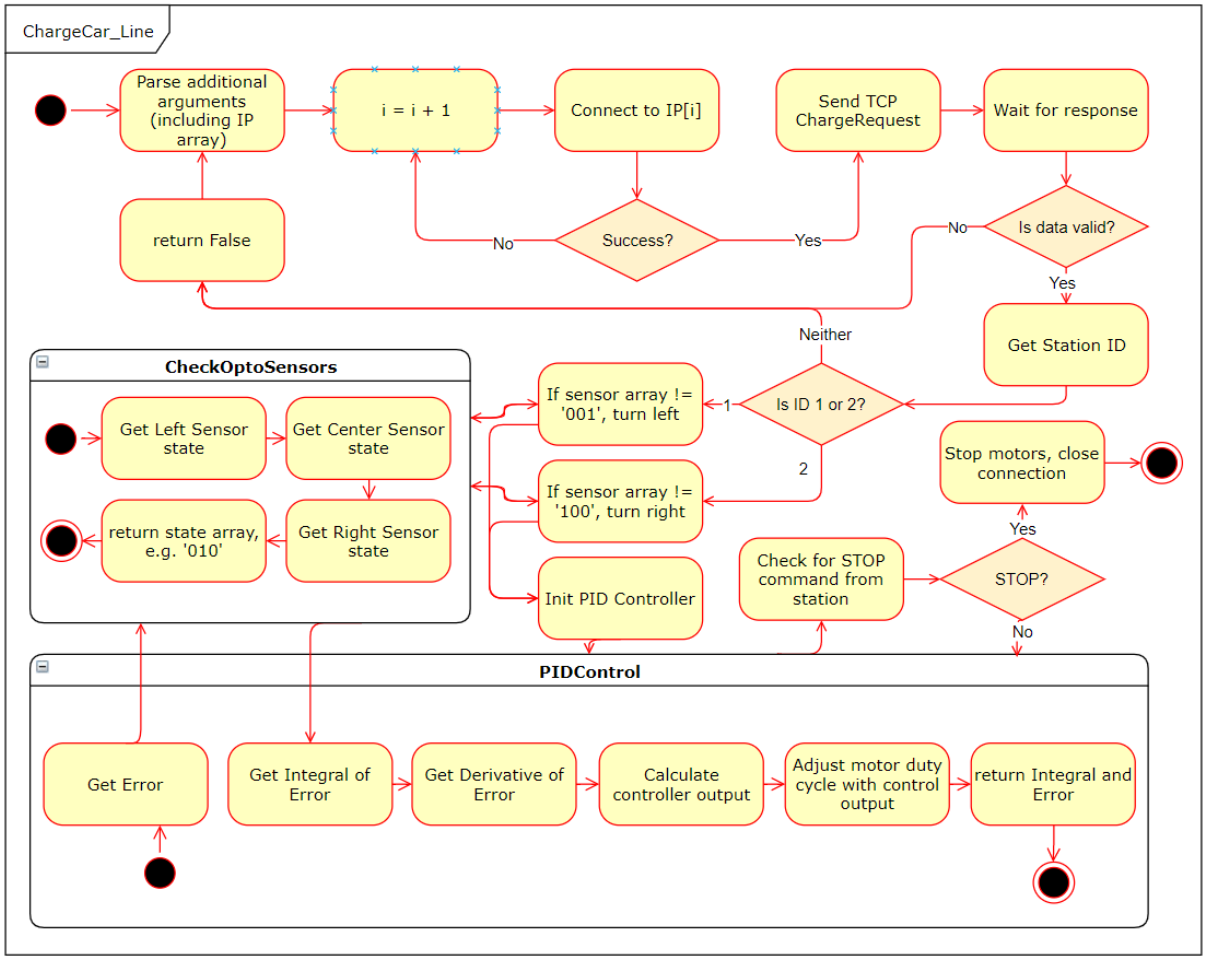


Figure 30: UML state machine diagram of the robot vehicle script.

The purpose of this script is to allow the robot vehicle to retrieve information about all the charging stations' status, and navigate to a free charging station autonomously for wireless power transfer.

The script accepts an arbitrary amount of IP addresses and its own TCP port as additional arguments. When the script is run, the script sets up the vehicle as a TCP client and attempts to connect to each specified IP address representing the charging stations in the network. If a connection to the first specified IP is unsuccessful, the vehicle attempts to connected to the second IP address, and so on.

If the attempted connection is successful, the script then sends a "ChargeRequest" message to the station server. After successful transmission of the request, the vehicle client should receive the free charging station's ID number as a response.

Depending on the ID number received, the robot vehicle will adjust its direction of travel to either the left or right side of the black line, if it is placed perpendicular to the line in such a way that all sensors are pointing at a black colored surface. After the vehicle has adjusted its direction, it initializes and runs a PID controller algorithm to follow the line.

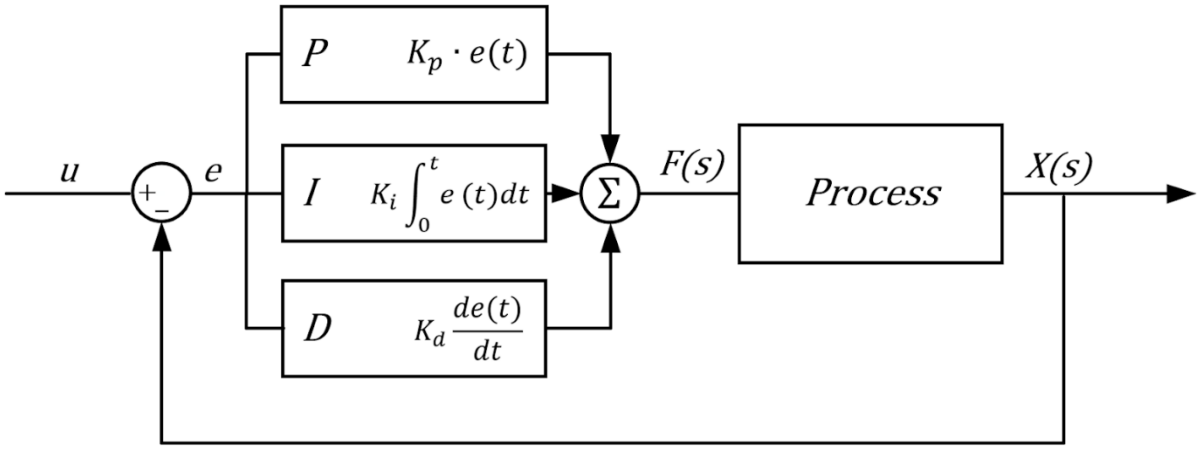


Figure 31: Block diagram of a classic PID controller. Source: https://dewesoft.pro/online/course/pid-control

For more information on PID controllers and their implementation, see References. To sum up, the controller is applied the input *u*, returns the output *X* and retrieves an "error" value *e*, which is equal to the difference between desired and actual output. This error value is used in calculating proportional, integral and derivative control terms according to preset values *Kp*, *Ki* and *Kd* in subsequent negative feedback loops, which are then used to correct the real output to be as close to the desired output as possible.

In this project, the PID controller algorithm retrieves its error values from the line sensor module readings:

```
value = CheckOptoSensors()
if(value == '000'):
      error = 0
elif(value == '001'):
      error = 15
elif(value == '010'):
      error = 0
elif(value == '011'):
      error = 5
elif(value == '100'):
      error = -15
elif(value == '101'):
      error = 0
elif(value == '110'):
      error = -5
elif(value == '111'):
      error = 0
return error
```

The error can then be adjusted with an offset, if desired. Once the error is retrieved, the integral and derivative values are calculated:

dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
error = GetPIDError() - offset
integral = lastintegral + error
if(integral > int_max):
     integral = int_max
elif(integral < int_min):
     integral = int_min
derivative = (error - lasterror)
```

The controller adjusts the vehicle's movement by modifying the motor duty cycles according to the calculated control output:

```
control = Kp * error + Ki * integral + Kd * derivative
changeMotorDC(speed + control, speed - control)
```

The *Kp*, *Ki* and *Kd* values are calculated using the Ziegler-Nichols method, a widely used method for tuning PID controllers. To tune a PID controller with this method, the integral and derivative control terms were disabled (set to 0), then the proportional control term was set to a level where the robot vehicle was consistently oscillating when following the line. This proportional gain is known is *Ku* (ultimate gain), and the oscillation period of the vehicle is referred to as *Tu*.

By default, *Ku* is equal to 400 and *Tu* is set to 425. However, they can be changed by passing them as optional arguments to the script, allowing for PID tuning without modifying the code itself. If the correct values of *Ku* and *Tu* are determined, the calculations for *Kp*, *Ki* and *Kd* are set to:

```
Kp = 0.2 * Ku
Ki = (0.4 * Ku) / Tu
Kd = (Ku * Tu) / 15
```

After each PID control loop, the script checks if the station server has sent a command to stop the vehicle. If no stop command is received, the PID control loop continues, however if a stop command has been detected, the motors are immediately stopped. The robot should then be positioned right above the station's transmitter coil, assuming the FSR is placed correctly, and the wireless charging process can begin.

For the full vehicle script, see "Car_script.py" in the report appendix.

3) Charging station code

The flowchart below demonstrates the main algorithm loop of the charging stations:

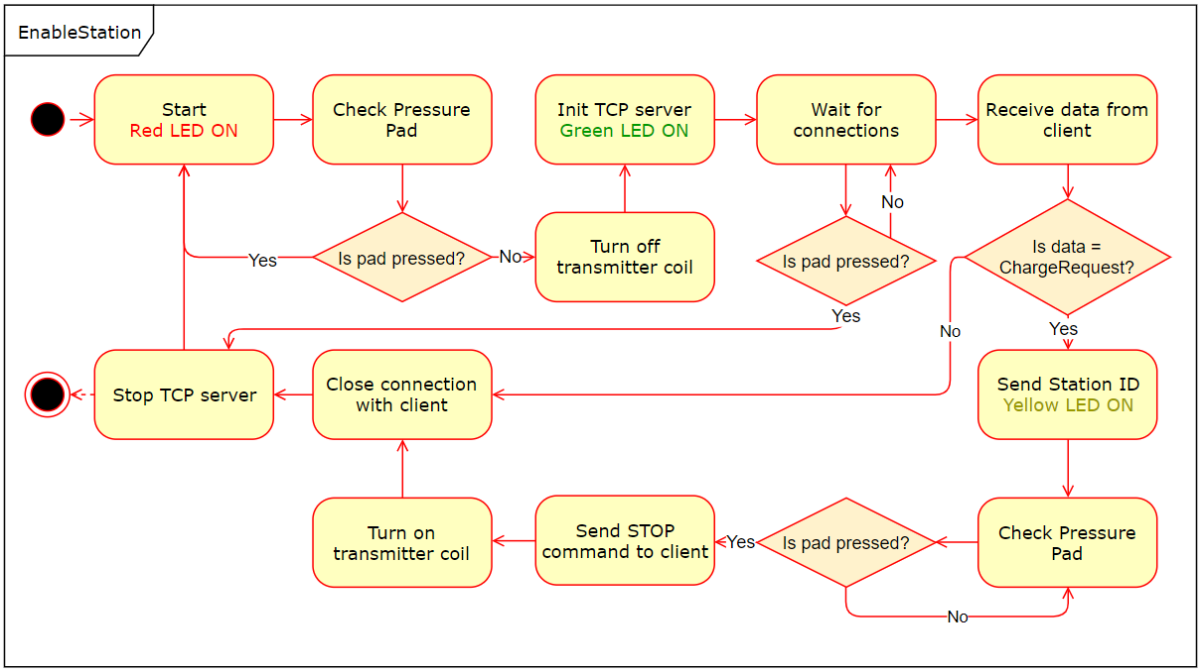


Figure 32: UML state machine diagram of the charging station script loop.

The purpose of this script is to allow the charging stations to retrieve information about whether or not it is being used or not, and communicate with the robot vehicle to help navigate it to itself for wireless charging.

The value that the FSR sends to the station determines its status - if there is no pressure applied to it, the station is free, and if there is pressure, the station is considered used. Before the TCP server is initialized, the status of the pad is checked first:

```
if (GPIO.input(PSInput) == GPIO.LOW):
      return True
else:
      return False
```

If/when the pad is no longer pressed, the TCP server is initialized using the inbuilt Python `socket` module:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(('0.0.0.0', TCP_PORT))
s.listen(1)
```

The TCP server is now initialized and listening for charge requests as long as the FSR is left unpressed. If the FSR is pressed again during this period, the TCP server shuts down and the script starts over again.

Once the station server establishes a connection with the robot vehicle (client) and receives a "ChargeRequest" message, it responds to the client by sending its designated ID number. After this point the station is considered "reserved", and is waiting for the robot vehicle to arrive.

When the robot vehicle arrives at the station, it presses on the FSR with its wheels, which causes the station to send a stop command to the vehicle client. This results in the robot vehicle stopping almost instantly after the station's FSR is triggered. The TCP connection is stopped, the transmitter coil is toggled on for enabling wireless power transfer to the robot vehicle, and the script starts over again.

For the full charging station script, see "Station_script.py" in the report appendix.

3. System overview

The following block diagram illustrates all interactions between the system's hardware components:

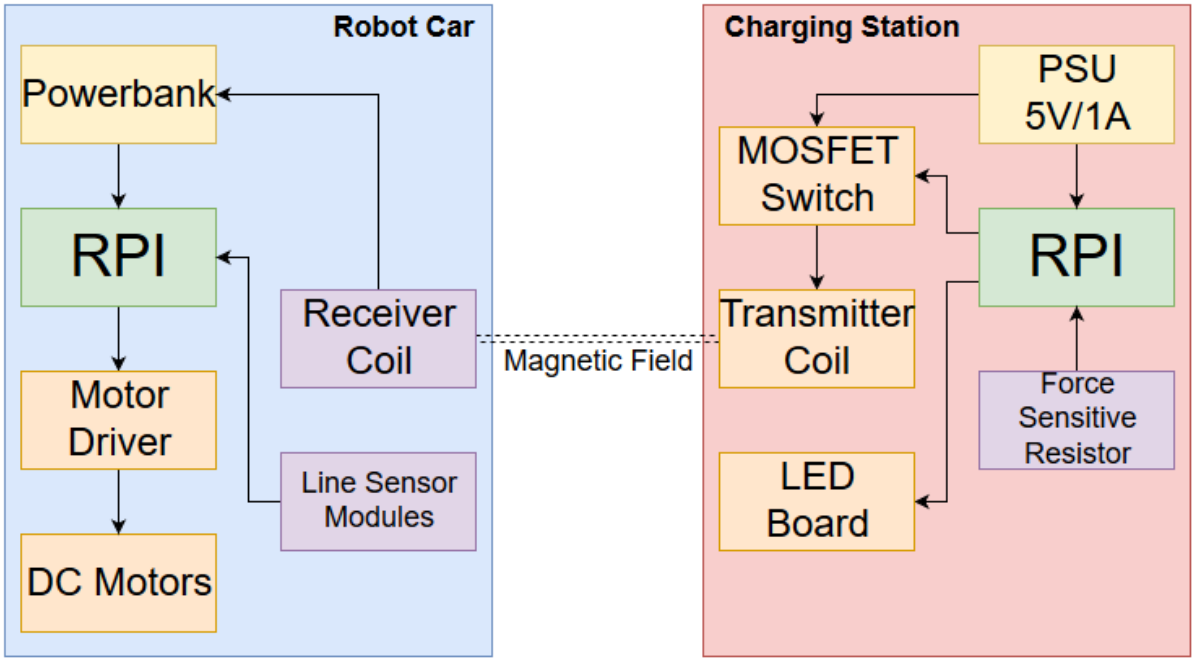


Figure 33: Block diagram of the system's hardware.

The main control units for both the robot vehicle and the charging stations are Raspberry Pis.

The motors and the Raspberry of the robot vehicle are powered by the Trust powerbank, which has the wireless receiver coil connected to its micro-USB input port for charging. The robot vehicle views its surrounding environment with three optical line sensor modules, and is able to navigate it using its four DC motors, which are controlled by the motor driver connected to the Raspberry.

The Raspberries of the charging stations are powered by 5V/1A power supplies connected to wall outlets. The charging station uses the FSR to determine its status (whether it is being used or not),

and controls the connected wireless transmitter coil using a MOSFET switch. It also has an LED array for indicating its status to external observers.

The diagram below demonstrates the complete functionality and timing of the system's software as a whole:

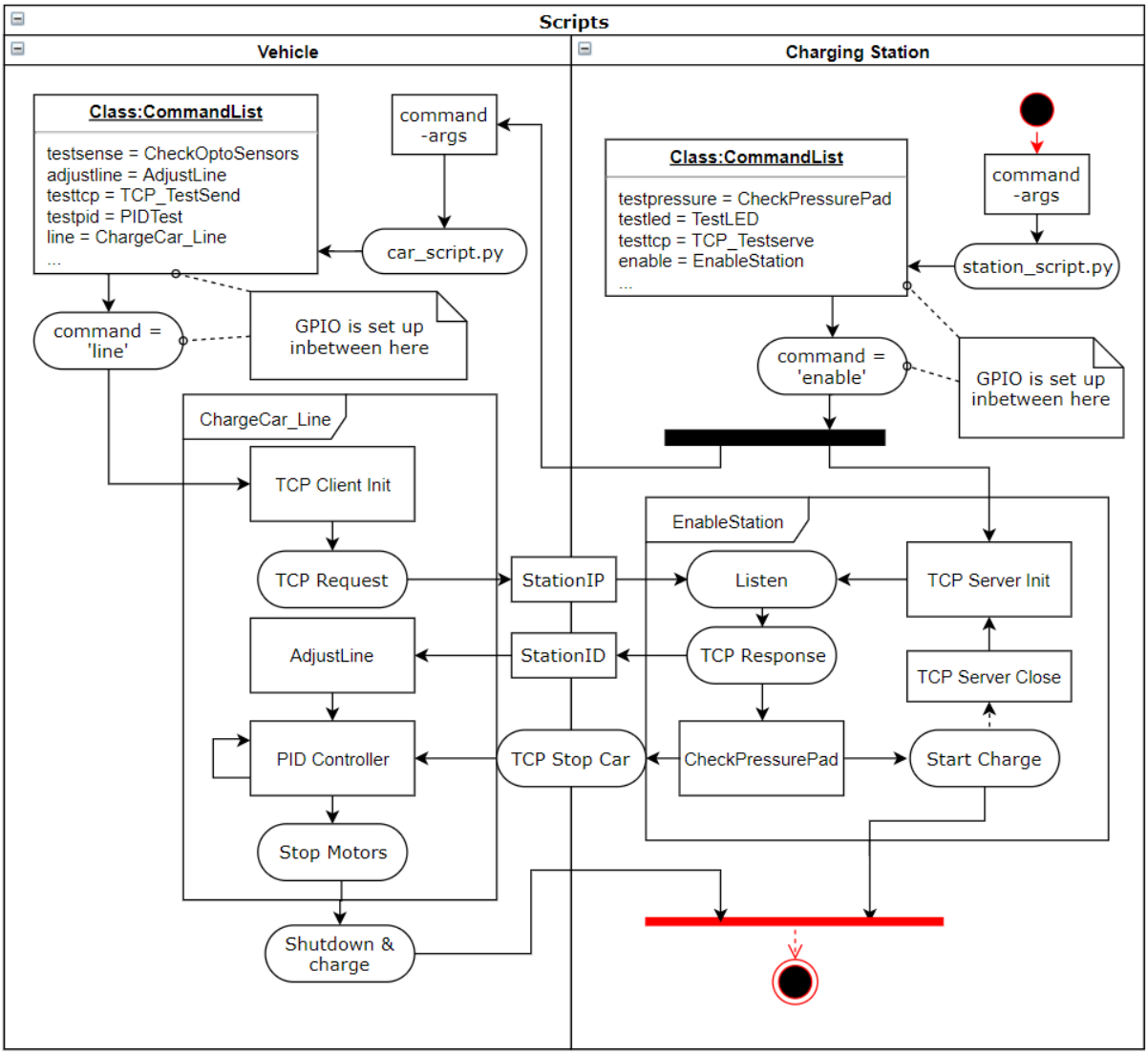


Figure 34: UML interaction overview diagram of the system's software.

The station script is launched first, with the 'enable' command passed as an argument to start the correct function. Once the station is prepared, the vehicle script is launched with the 'line' command and all IP addresses of the stations as arguments.

The vehicle script has additional functions for PID, TCP client, sensor and motor testing, in addition to manual driving mode. The charging station script has additional functions for LED, coil toggle, FSR and TCP server testing.

The vehicle script accepts the station IP array & TCP port as the required arguments. In addition, it accepts desired motor duty cycle, PID error offset, PWM frequency, Ku and Tu as optional arguments.

The charging station script accepts its designated ID as a required argument. In addition, it accepts a custom TCP server port as an optional argument.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

IV. Results

This chapter demonstrates the results of the project's development process, some of the problems encountered and solved during development, and some thoughts and recommendations for future works on this system.

1. Current state

Theoretically, all the project goals have been achieved successfully - the system has the hardware and software components needed to be functional according to initial designs. In practice, due to time constraints, the system design and build is not entirely optimized, though it is able to function as intended under certain conditions.

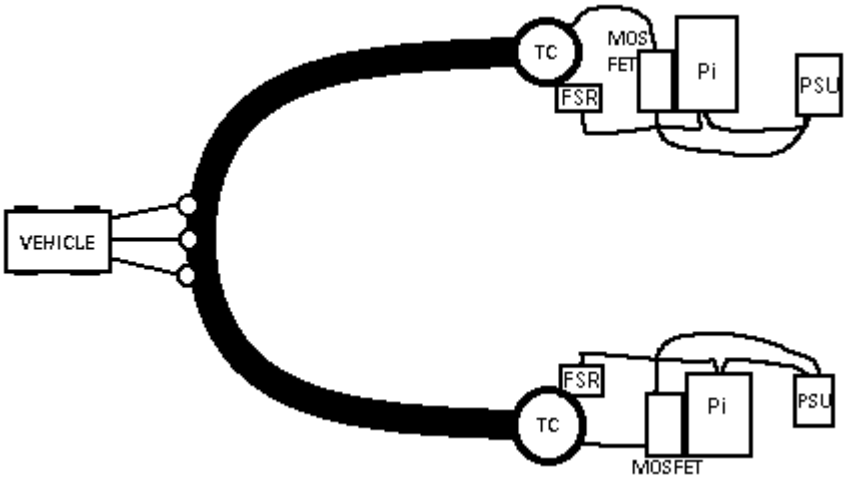


Figure 35: Sketch of the system design.

The hardware and software for the robot vehicle has been designed, built and tested successfully.

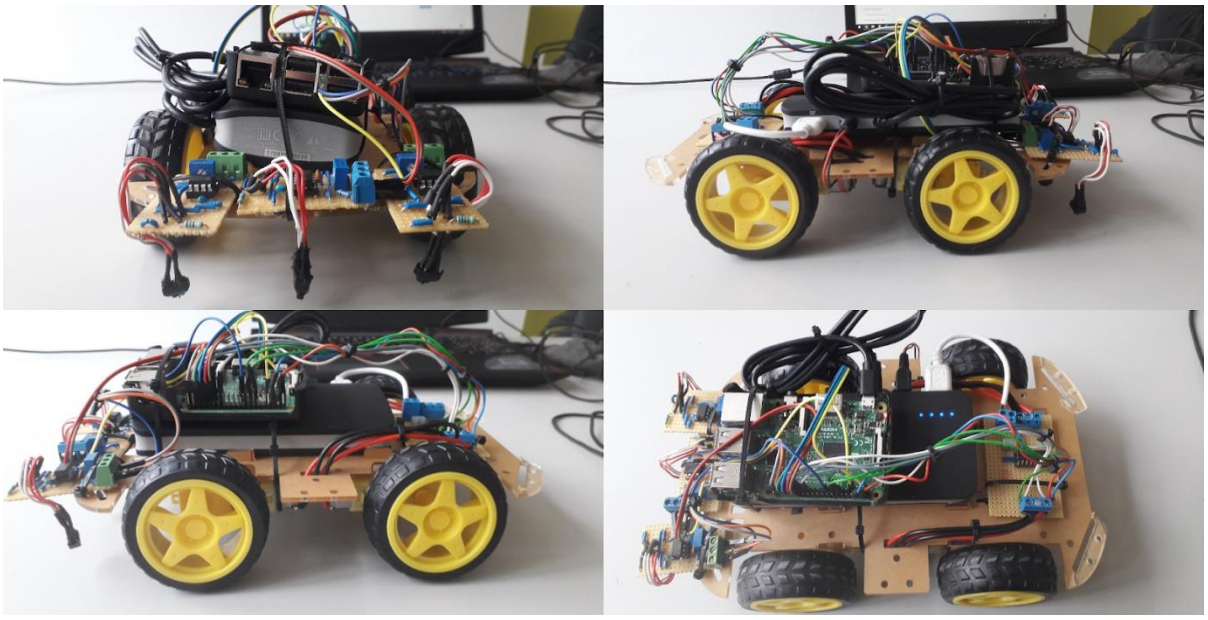


Figure 36: The robot vehicle assembly.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

In addition, there were two “smart” wireless charging stations designed and built. The software for the charging stations is complete, however some of the hardware components were missing at the time of writing due to their delayed delivery dates (for more information, see the Log Book in the report appendix). The stations are expected to be fully finished soon.

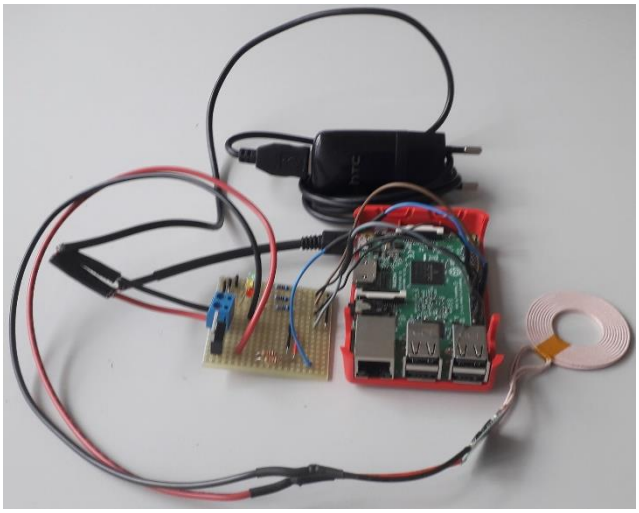


Figure 37: A “smart” wireless charging station.

In the current state the following tests were done:

- Testing the line sensor modules on the robot vehicle. All the sensor modules pass the correct values to the Raspberry as long as they are less than about 1.8 cm away from the surface measured.
- Testing the pressure pad on the charging stations. On both stations, the pressure pad passed the correct value to the Raspberry Pi, without any recorded false positives.
- Testing the motors on the robot vehicle. The Raspberry was able to change direction and speed of both motors easily, allowing the vehicle to drive forward, backward and turn left/right, with different motor duty cycles.
- Testing the robot vehicle PID controller line following algorithm on a circuit “race” track. After about 50 successful consecutive laps, it was deemed to be tuned well enough.
- Testing TCP/IP communication between vehicle and stations. The stations were set up as servers waiting for data transmissions, which were echoed back to the client. The client (robot vehicle) sent arbitrary sized data to the servers and had received it back successfully.
- Testing the LED indicators on the stations. All the LEDs lit up at the correct times.
- Testing the transmitter coil toggle functionality on the stations. The transmitter coil did not get enough current to be able to charge the vehicle’s powerbank due to a separate issue, however the MOSFET switch performed its function correctly by disabling/enabling current flow to the coil with an output signal from a Raspberry GPIO pin.
- Testing the main system - having the robot vehicle find a free charging station and navigate to it autonomously. All individual components of the system worked as intended, however as a whole the tests saw mixed success. The main hindrances to reliable functionality are the

robot vehicle direction adjustments before initializing the PID controller, and the stations' unoptimized design (where the coil is placed, where the FSR is located, etc.)

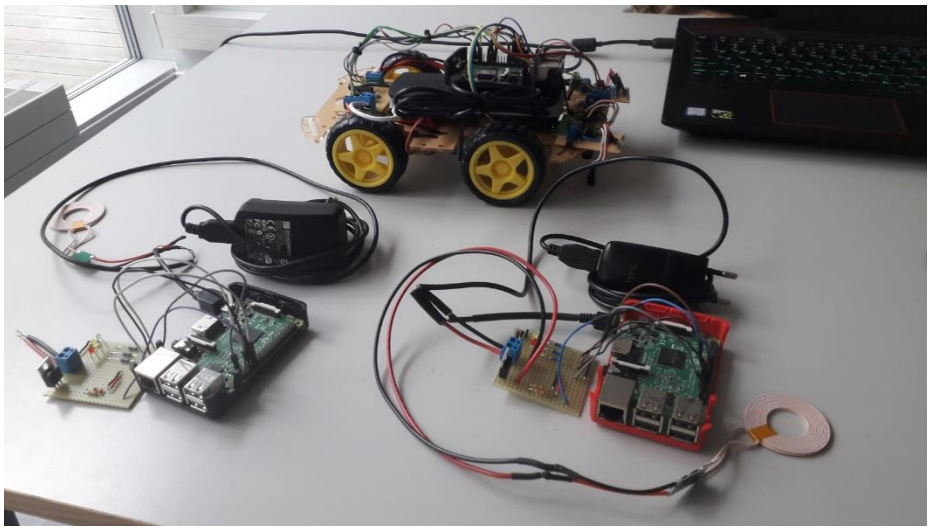


Figure 38: All components of the project.

Overall, the project is deemed to be an almost complete success, since it is feature complete and only needs a few small tweaks to function as intended reliably.

2. Problems and solutions

This section describes the various problems encountered during the project's development process, along with discovered or potential solutions to them.

Problem: How can the wireless charging station determine whether it is currently used by the robot vehicle?
Solution: This problem was solved by using a force sensitive resistor (FSR) which can detect if any pressure is applied to it. The FSR is connected to the Raspberry GPIO and placed near the transmitter coil in such a way that it is triggered by the wheels of the incoming robot vehicle, informing the station that the vehicle has arrived and is attempting to charge itself.

Problem: How is the robot vehicle able to determine if a specific charging station is free to be used or not?
Solution: To solve this problem, the robot vehicle and the charging station must be able to communicate wirelessly in some form. This was achieved by using TCP/IP communication between the vehicle and the station, with Python scripting. The charging stations are set up as TCP servers and have their own unique IP addresses, which can be passed to the vehicle script so that the vehicle can check to see if the station is used or not. The vehicle attempts to connect to each server, and if a connection is successful a request for the station's status is sent, and a TCP response is received.

Problem: How can the robot vehicle determine where the charging station is? How does it navigate itself to the station?

Solution: Due to time and budget constraints, the project needed a cheap and quickly achievable solution to this problem. In the end, the chosen method for navigating the vehicle to the station was to draw a line between the vehicle's initial position and the charging station, and have the vehicle follow the line, delivering itself to the correct spot for charging. The line following was achieved by designing a PID controller algorithm and implementing it in the script, ran by the Raspberry in the robot vehicle.

Problem: The powerbank seems to reduce output voltage significantly when it is being charged, which results in the robot vehicle's Raspberry Pi and motors shutting down.

Figure 39: The output voltage to the Raspberry Pi and motors from the powerbank, measured when it is not being charged.

Figure 40: The output voltage, measured when it is being charged.

Solution: In order to solve this problem, a potential solution was implementing the ability for the charging station's Raspberry to control the current flow of the transmitter coil. This way, the charging station can stop the robot vehicle at the correct spot without the system behaving unpredictably, which would mitigate the problem. However, the transmitter coil needs 5V/1A to function properly, and this amount of power cannot be delivered by the Raspberry GPIO as it is far too high for it to handle. This resulted in the creation and design of the MOSFET switch, as it allows for a tiny amount of current from the Raspberry GPIO to control the flow of massive amounts of current needed for the transmitter coil, without risking damage to the Pi.

Problem: The MOSFET switch works, but the 5V/1A AC/DC wall socket power supply is unable to supply both the Raspberry and the transmitter coil in the charging station. The transmitter coil is only getting ~3.6 V after it has been routed through the MOSFET, which is not enough to supply the powerbank connected to the robot vehicle.
Solution: A potential solution to this problem is changing the wall socket power supply to support increased current flow, up to about 2 A, providing both the transmitter coil and the Raspberry with sufficient power for intended operation. Care should be taken not to overload the components with too much current and causing damage.

3. Discussion and recommendations

The final project development period was officially started in the middle of April 2019, with the deadline being roughly the start of June 2019. The total development time is therefore about one and a half months, which is significantly shorter than previous project development periods done in the IT Technology education.

This time constraint proved to be the greatest hindrance to the project's development goals and workflow, even more so than the limited budget. As a result, many of the project's goals and ambitions had to be scaled down and simplified, while the solutions to various problems encountered during development were chosen based on how fast they could be implemented. To keep up the constant progress needed for completing the project before the deadline, the quality of the development results has arguably suffered consequently. Still, the team behind the development process believes that this is a better alternative than keeping their initial ambitions and likely failing to complete the project in the designated time frame. In the end, the development process resulted in a system that more or less completes all the initially set objectives.

To name a few design decisions done to keep the development as agile as possible:

- The Raspberry Pi being the control unit for both the robot vehicle and the charging stations. Its processing power and integrated Wi-Fi module are incredibly useful and easy to set up, however its GPIO is not well suited to timing critical applications, such as the PID controller or the TCP/IP communications at specific intervals. For these use cases, a more suitable control unit would be something similar to an Arduino Uno or a different ARM Cortex-M based board, like the Tiva board.
- The charging station determining its own status by using a pressure pad. A more elegant (and time-consuming) solution could be usage of a different type of sensor, such as a distance measuring ultrasonic sensor, or determining presence of the robot vehicle by measuring the changing magnetic field of the wireless transmitter coil.
- The robot vehicle locating the charging station by following a line, and choosing between stations by adjusting itself left or right. This solution is unscalable, unreliable and generally unsuitable for any real use case beyond very niche applications. However, the only real alternative solution to this problem is some form of computer vision or other type of environment detection, which can be made into a different project in itself with how much time and solution architecture it would require to implement.

In addition, the project development experienced the more usual difficulties such as long component arrival times and limited development team size.

On the bright side, despite the short development time, some impressive results were achieved by the project team, mainly due to their experience in electronics & software development, as well as proper time & resource management. Constant progress was made and milestones were achieved due to avoidance of common pitfalls similar projects had in the past, such as scope/feature creep, emphasis on theoretical progress over practical outcome, and lack of cohesion between team members. As a result, the project is designed to be as modular and scalable as possible, allowing for further development potential.

Some features that could be done in future work:

- Better alternative solutions mentioned above to replace the current ones
- Implementation of current measurement in the robot vehicle, allowing for a custom battery level indicator
- PCB designs, builds and proper casings for both the robot vehicle and charging stations
- Improving the motors and motor driver for the robot vehicle

dain0084@edu.eal.dk
stef1571@edu.eal.dk

V. Conclusion

1: A proof of concept for the wireless charging solution has been successfully built and tested. It proved that a wireless charging solution for the robot vehicle is feasible.

2: An autonomous robot vehicle was designed, built, programmed and finished successfully. It can follow lines on its own, and can also be remotely driven with commands from external inputs.

3: Two wireless charging stations capable of diagnosing their own status and sending/receiving data were successfully built. The wireless charging modules were tested and were working as intended.

4. A system where the robot vehicle can identify and drive itself to a free charging station was built and tested. In theory, it is fully functional and has all the features needed to achieve this objective, however in practice it lacks reliability and elegance due to some design flaws. If more time was available, this system could be fixed and even improved and/or supplemented with additional designs.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

VI. References

1. Bibliography

PhD, Renato M.E. Sabbatini. "Sabbatini, RME: An Imitation of Life: The First Robots". Retrieved from http://www.cerebromente.org.br/n09/historia/turtles_i.htm at May 16, 2019.

Calvert, James (2001). "Inside Transformers". University of Denver. Retrieved from https://web.archive.org/web/20070509111407/http://www.du.edu/~jcalvert/tech/transfor.htm at May 19, 2019.

Lee, Reuben. "Air-Core Transformers". Electronic Transformers and Circuits. Retrieved from http://www.vias.org/eltransformers/lee_electronic_transformers_07b_22.html at May 18, 2019.

Steve Winder. “Power Supplies for LED Driving (Second Edition)”, 2017. Pages 241-247.

Ahmed Gamil, Franz Schatzl. “Determining transformer core losses”, 2015. Retrieved from https://hrcak.srce.hr/file/268192 at May 20, 2019.

Prasad Umesh Kumar. “Wireless Power Transmisson using Resonant Inductive Coupling”. Retrieved from https://www.ijraset.com/fileserve.php?FID=15455 at May 22, 2019.

Microstar Laboratories. “Ziegler-Nichols Tuning Rules for PID”. Retrieved from http://www.mstarlabs.com/control/znrule.html at June 1, 2019.

2. Useful links

Coil inductance calculator:
https://www.allaboutcircuits.com/tools/coil-inductance-calculator/

More about other wireless charging types and standards:
https://batteryuniversity.com/learn/article/charging_without_wires
https://www.androidauthority.com/wireless-charging-qi-pad-technology-580015

More about mutual inductance:
http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/indmut.html

More about the Q factor:
https://www.electronics-notes.com/articles/basic_concepts/q-quality-factor/basics-tutorial-formula.php

2N4401 datasheet:
http://www.farnell.com/datasheets/661741.pdf

dain0084@edu.eal.dk
stef1571@edu.eal.dk

Raspberry Pi electrical specifications (unofficial):
http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications

LM293 datasheet:
https://www.ti.com/lit/ds/symlink/l293.pdf

TCRT5000 datasheet:
https://www.vishay.com/docs/83760/tcrt5000.pdf

Connecting to a Raspberry Pi with an Ethernet cable:
https://www.instructables.com/id/Intro-to-the-Headless-Raspberry-Pi/

Datasheets of the wireless charging module ICs:
http://datasheetcafe.databank.netdna-cdn.com/wp-content/uploads/2017/03/XKT-412-datasheet.pdf
http://www.datasheetgo.com/t5336-sop-8-pin/
http://aitendo3.sakura.ne.jp/aitendo_data/product_img/ic/power/T3138/T3138.pdf

IRL1004 power MOSFET datasheet:
http://www.irf.com/product-info/datasheets/data/irl1004.pdf

DFRobot and Seeed Studios main website:
https://www.dfrobot.com/
https://www.seeedstudio.com/

PID controller implementation:
https://dewesoft.pro/online/course/pid-control

Example Python code for TCP communication:
https://wiki.python.org/moin/TcpCommunication

VII. Appendix

1. Project logbook – Multiple Wireless Charging Stations – UCL – Stefan & Dainius

Date: March 28 & 29, 2019 **PROJECT INITIATION**

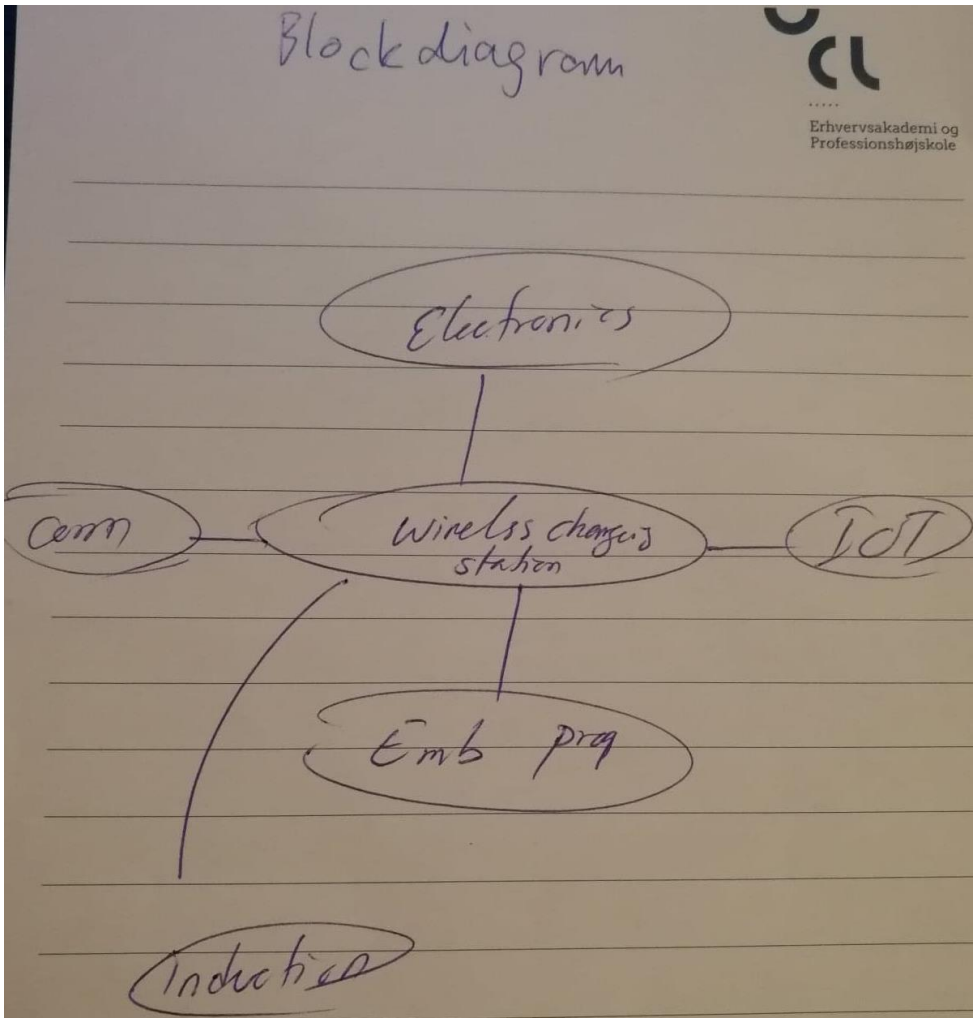
We met up with our project supervisor Ilias Esmati to talk about it and find out what our possibilities were.

We found a project for our last thesis. The project will be about wireless charging of robot cars. The title of the project is “Multiple Wireless Charging Stations". While working on our last thesis, we will also teach/supervise the second semester students on our education, since they are making a similar project in the electronics laboratory.

The robot car should be able to go to a charging station by itself, furthermore it should be able to detect if the charging station is used by another device or if its free to use.

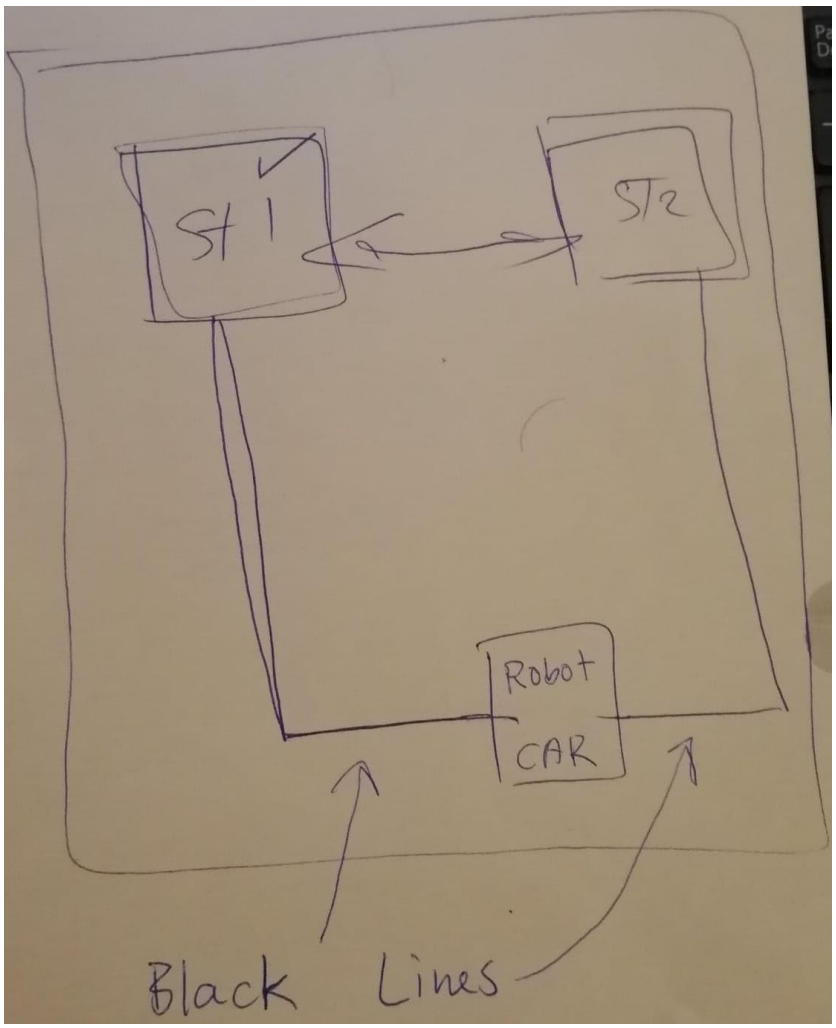
The next step for us is to make a project description and a time plan of our project. Furthermore, we must plan for what to teach/assign to the second semester students.

On the first picture you can see some of the subjects and theories which will be included in our final project. This is an early block diagram of our system.

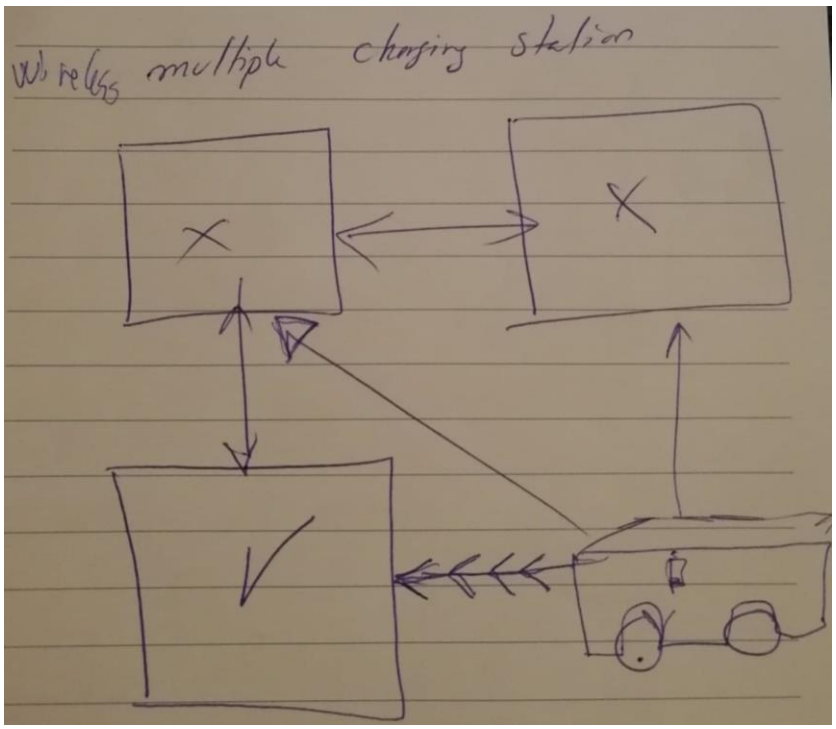


Second picture shows an example of how the robot car could navigate to a charging station.

dain0084@edu.eal.dk
stef1571@edu.eal.dk



The last picture shows an example of the robot car checking which charging station is free to be used.



dain0084@edu.eal.dk
stef1571@edu.eal.dk

Date: April 1-5, 2019 **RESEARCH**

In this week, we have been researching on our project. Firstly, we have been looking for companies who deliver wireless power solutions for robots. In this search, we did not find many companies, doing this kind of systems.

The first company we found is called WiBotic[1], which is based in The United States. This company specialises in wireless charging solutions for drones and robot cars.

Next company is called VOLTERIO[2], and this company is based in Austria. They specialize in wireless charging solutions for electric cars.

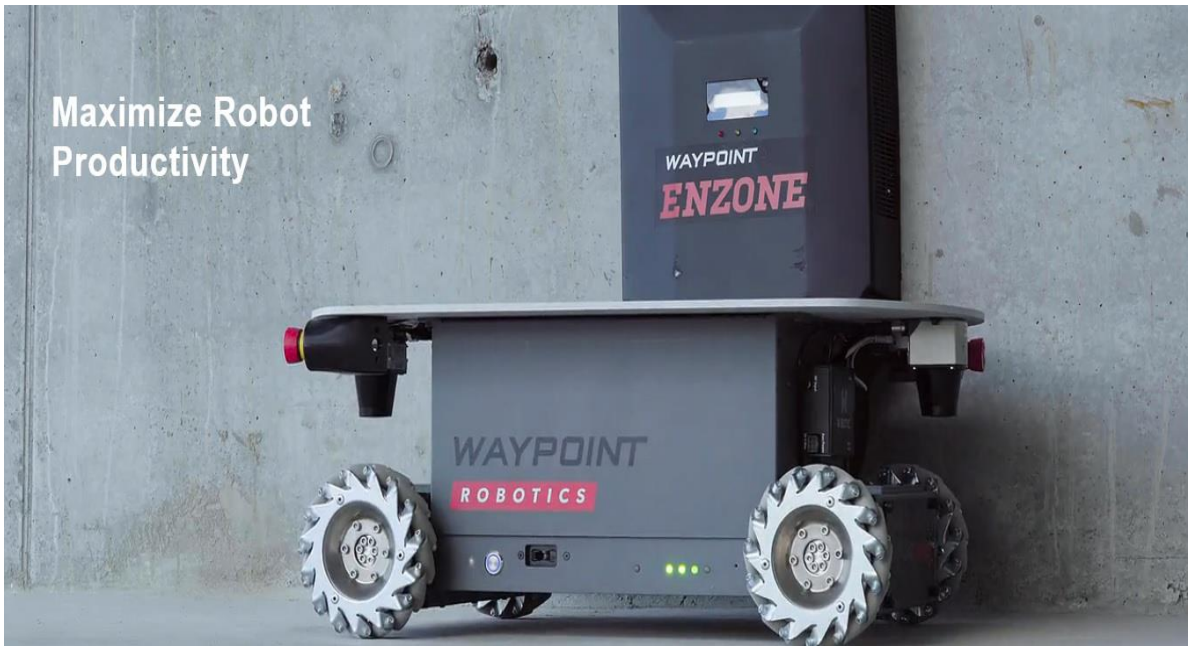
The third company is called Plugless[3] and like VOLTERIO, they specialize in wireless charging of electric vehicles. This company is in The United States of America.

The last company we found is called jjPlus[4] and they are based in Taiwan. They provide wireless charging for a wide number of products, from robots, to drones, laptops, phones and so on.

So, on the world basis there are not many companies doing this kind of things and if we make a great final project with a great product, we could potentially open our own business after we graduate.

Furthermore, we started working on our project description, time plan and block diagram, so we can meet up with our project supervisor to check with him and see if its good or it can be bettered.

The picture below shows a wireless charging system made by WiBotic, which is charging a robot car.



Links:

The Truth About Wireless Charging - YouTube
How Does Wireless Charging Work? || Crude Wireless Energy Transfer Circuit - YouTube
Mini Project: DIY Wireless Charging - YouTube

Date: April 8, 2019

[1] https://www.wibotic.com/
[2] http://www.volterio.com/
[3] https://www.pluglesspower.com/
[4] http://www.jjplus.com/

dain0084@edu.eal.dk
stef1571@edu.eal.dk

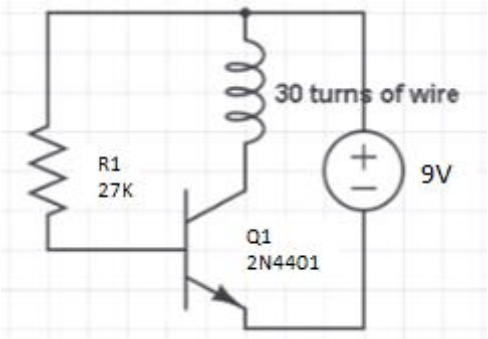
We continued the work on the project description, block diagram and time plan, so we can meet with our supervisor Thursday and discuss it with him.

Date: April 9 & 10, 2019
These two days we made the project description, the block diagram and time plan. Furthermore, we started on a parts list of the needed components in our project.
We have sent the project description, time plan, block diagram and parts list to our supervisor.

Date: April 11, 2019
Today we had a meeting with our supervisor regarding our project description, time plan and parts list. He told us that it all looked good but recommended that we put the prices of the components on the parts list also, so he can order them.
Furthermore, we must finetune our project description, time plan and block diagram, but it's a good beginning.

Date: April 12, 2019
The next couple of weeks we will be looking into the electronics and circuit designing of a wireless charging station, and assembling, wiring and programming a robot car with different sensors.
Links:
7 Types of Industrial Robot Sensors
Robot Sensors | Sensors for your Robot
2N2222AHow to Make Wireless Power Transmission - YouTube
2N3904Wireless electricity transmission circuit - YouTube
2N2222How To Make a Wireless Charger at Home - Easy Way - YouTube
2N2222How to Make Wireless Charger | Turn Your charger Wireless - YouTube

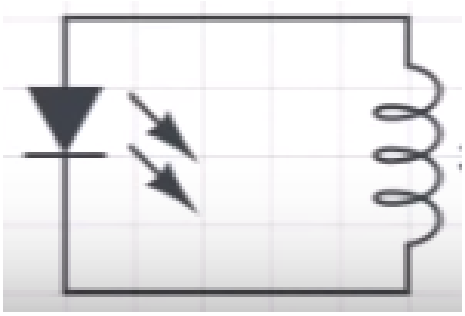
Date: April 15-19, 2019 **DEVELOPMENT**
This week we worked on making a wireless power transfer in the lab. For this we needed two coils made of enamelled copper wire, a resistor, a transistor, wires and a breadboard or veroboard.
First, we took some copper wire and started making the transmitter coil. We took a soda can and made 15 turns and a loop and 15 more turns. For the receiver coil we made 30 turns and then that coil was ready. After that we scraped of the insulation on the terminals of the two coils.
Next thing we made was the transmitter circuit and receiver circuit. Here is a picture of the schematic[5] of the transmitter circuit.



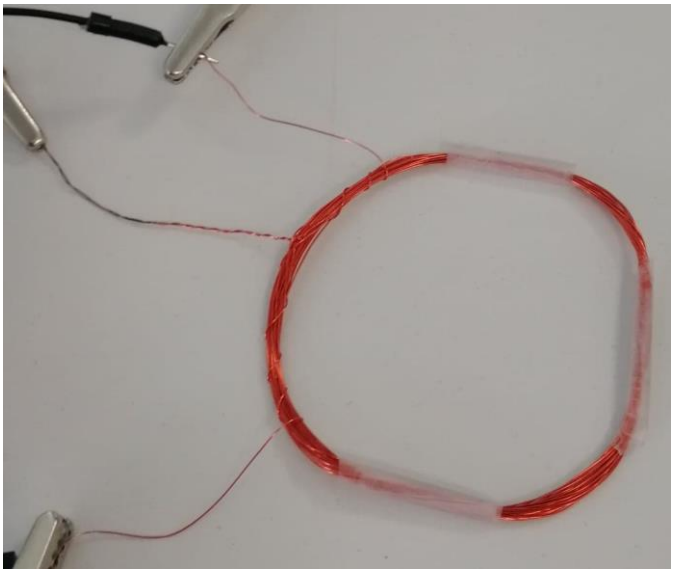
[5] https://www.youtube.com/watch?v=-7DgPmkg-74

dain0084@edu.eal.dk
stef1571@edu.eal.dk

On the picture we can see that we must use a resistor, transistor, a coil and a power source. In the beginning we could not make the circuit work properly, because of the 2N3904 transistor, it could not handle the current it received. The max current it can handle is 200mA.
The next thing to do now is to find alternative transistors we can use instead. Here we found three other replacements. Firstly, the 2N4401 which can handle 600mA, secondly the PN2222 which can handle a current of 600mA also and lastly the 2N2222 which is rated for 800mA. we checked the E-Lab to see if we could find some of these transistors and luckily, we found the 2N4401 transistors. When testing it now we happily found out that everything worked perfectly, we were able to transfer power wirelessly between two coils. The receiver circuit consists of a coil and a LED as depicted below[6].



After that we measured how many milliamps the transmitter circuit uses when running at 5 and 9 volts. Here we found out when it receives 5 volts it is using 0,045 amps and when it runs at 9 volts it is using 0,100 amps.
Lastly, we can confirm that our proof of concept exercise works and its well documented, so the next we must do is to order the last parts and start the development of the system.
Under here is some pictures that describes some of the processes in making the proof of concept example with wireless transfer of power. The first picture shows the transmitter coil.

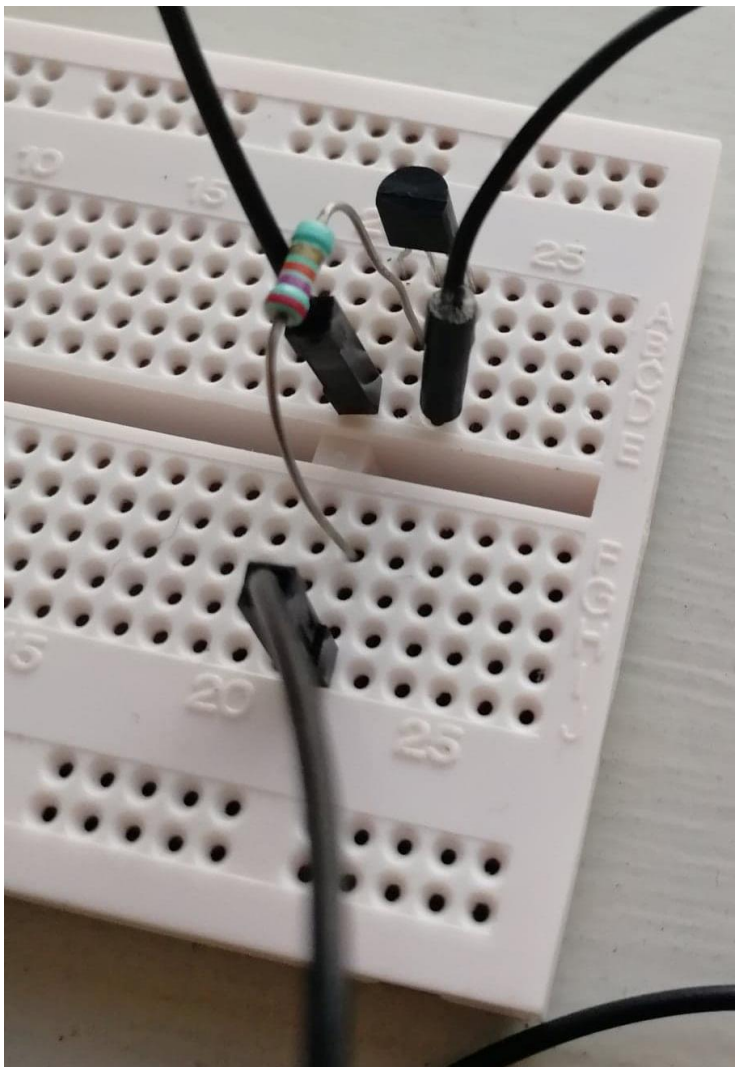


The next picture shows the receiver coil.

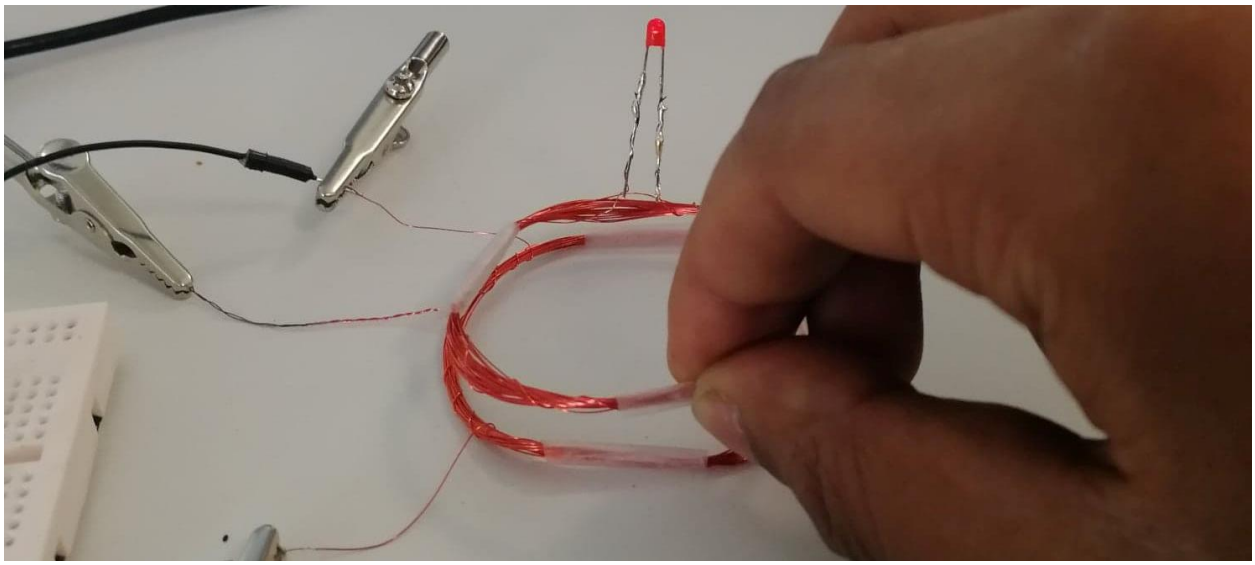
[6] https://www.youtube.com/watch?v=-7DgPmkg-74

dain0084@edu.eal.dk
stef1571@edu.eal.dk

Third picture shows the transmitter coil circuit on a breadboard.

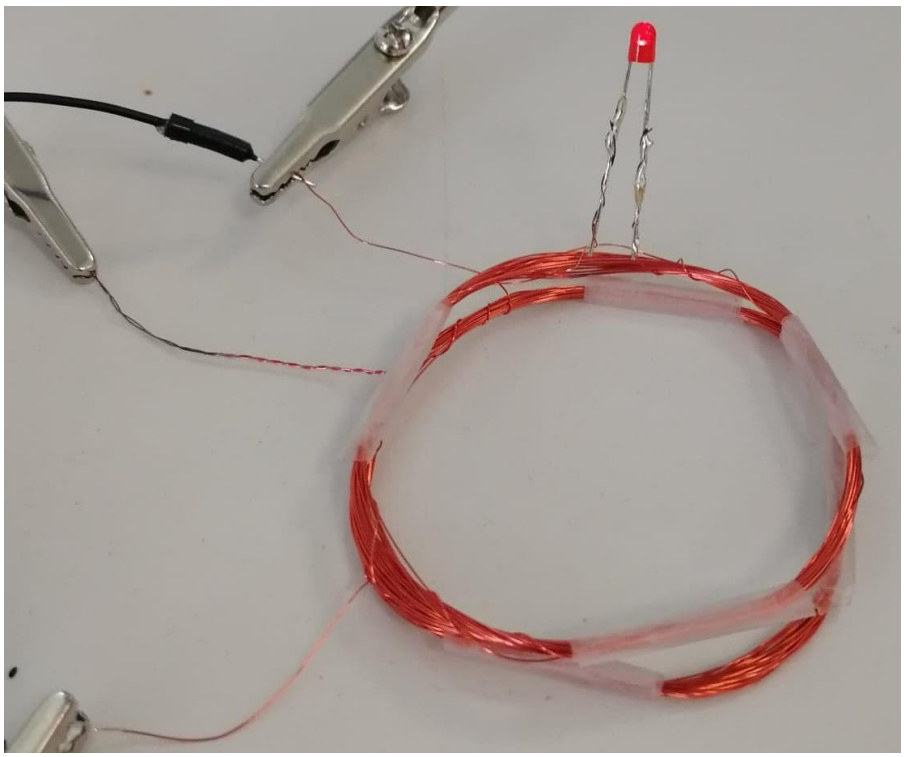


The fourth image shows the coils which are 1 centimetre away from each other and we can see a mild glowing LED.



The last image shows that the coils are very close, and the LED is glowing well. In this picture we made the proof of concept and made a wireless transmission of power.

dain0084@edu.eal.dk
stef1571@edu.eal.dk



Links:
2N2222AHow to Make Wireless Power Transmission - YouTube
2N3903 - General Purpose NPN Transistors - 2N3903-D.PDF
661741.pdf
PN2222 - General Purpose NPN Epitaxial Transistors - PN2222-D.PDF
211209_1.qxd - 296640.pdf
2N3904 vs 2N2222

Date: April 22-26, 2019

This week we did a lot work on the project, mostly on the robot car side. We also measured the internal resistance of our proof of concept coils. The internal resistance we measured was around 3-4 ohms.

We assembled the robot car base, made the motor driver[7] and line sensor boards[8]. First, we intended on making a robot car with two motors and one stabilizer wheel, but since we could not find any stabilizer wheel in the electronics lab, we then went with 4 motors instead of the intended two.

When testing the robot car base, we found out that the motor driver and all motors were working.

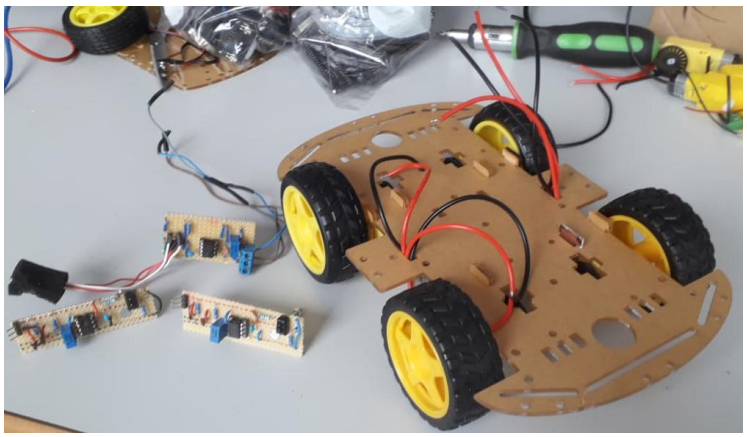
When testing the line sensor boards with the oscilloscope, we found out one of them wasn't working. We must do some troubleshooting of the nonworking line sensor board or maybe make a new one.

Furthermore, the last parts were ordered for the project, the parts are from within Europe, so we expect the delivery time to be short.

The picture below shows the robot car base, two line sensor boards and the motor driver board we made.

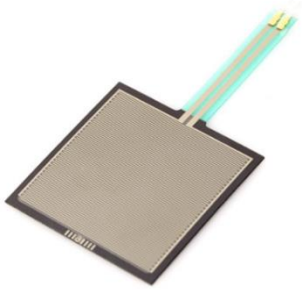
[7] https://eal-it-technology.github.io/RaspberryFun/hardware/motordriver/
[8] https://eal-it-technology.github.io/RaspberryFun/hardware/sensor/

dain0084@edu.eal.dk
stef1571@edu.eal.dk



Date: April/May 29-3, 2019

This week we used most of our energy and time on the internship report hand-in and internship poster presentation. This doesn't mean that we did not work on the project.
We started re-evaluating/researching which sensors we need for the robot car, after some discussions we settled on using the line sensors which we already have made and the force sensitive resistors which we have already ordered for our project.
Another thing we started on was to make a report mock-up, so we can start writing on the report right now, so we have less stress before the deadline.
Next thing we did regarding the project was to start planning and talking about the code and how to build up the code for our system.
Lastly, we initiated the final project report and began building the base of it and writing about different topics regarding our project.
The picture below shows the sensors which are going to work with our robot car.



Date: May 6-10, 2019

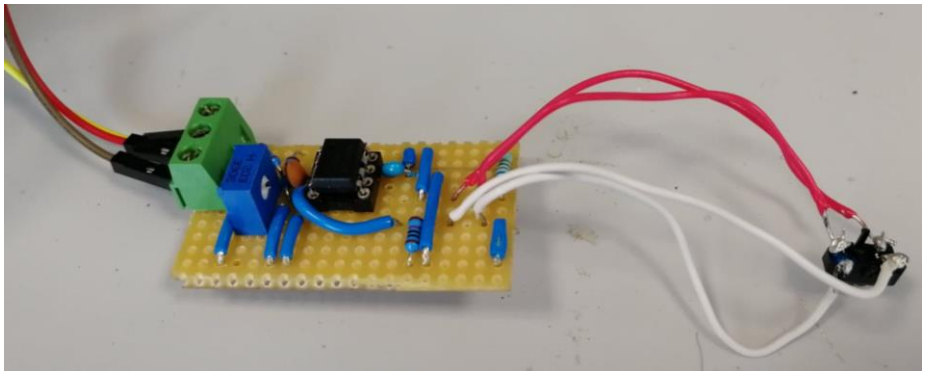
We have been talking about how to power our four motors on the robot car and our whole system generally. The transmitters will be powered through the wall outlets or the 5V pin on the Raspberry Pi. The Receiver or the motors and the Raspberry Pi on the robot car will be powered by a power bank. We must investigate how to power all four motors from the power bank itself.
Another thing we did this week was to set up all three Raspberry PIs for our system. One for the autonomous robot car and two for the wireless charging stations. First, we installed the latest version of Raspbian[9] on all the of the Raspberries. Then we enabled SSH[10] and lastly, we connected

[9] https://en.wikipedia.org/wiki/Raspbian
[10] https://www.raspberrypi.org/documentation/remote-access/ssh/

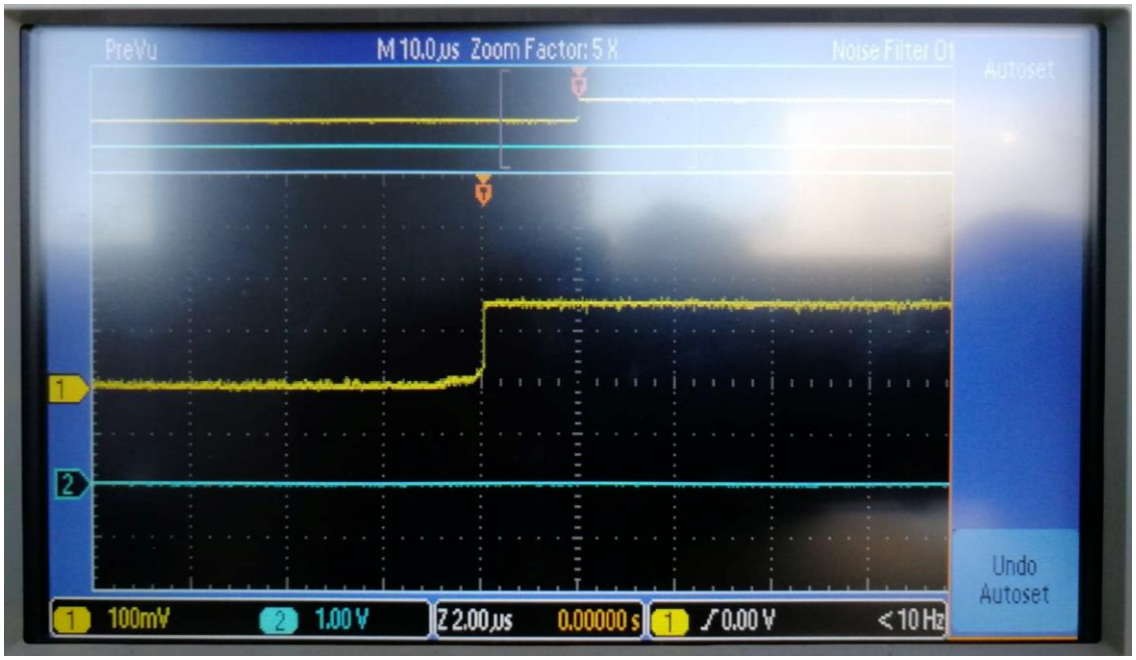
dain0084@edu.eal.dk
stef1571@edu.eal.dk

all the Raspberries on the same WIFI network. We are thinking about giving the PIs static IP[11] addresses.
Now that we have all the Raspberry PIs working and running on the same network, we can start thinking about which protocol to use for the PIs to communicate with each other. We have been considering on MQTT[12], TCP[13] and UDP[14], but we will decide on that later in the project.

Date: May 13-14, 2019
These two days we worked on the robot car and the nonworking line sensor board.
Regarding the robot car, we mounted a power bank with strips and slowly started on mounting the Raspberry PI, the motor driver, the sensors and wire it all together on top of the robot car base.
When we troubleshooted the nonworking line sensor board we could not figure out what were wrong with it. Luckily the electronics laboratory had all the components needed for making a new line sensor board, so that's what we did[15]. Now we have two working line sensor boards.
The picture below shows the new line sensor board we made in the E-Lab.



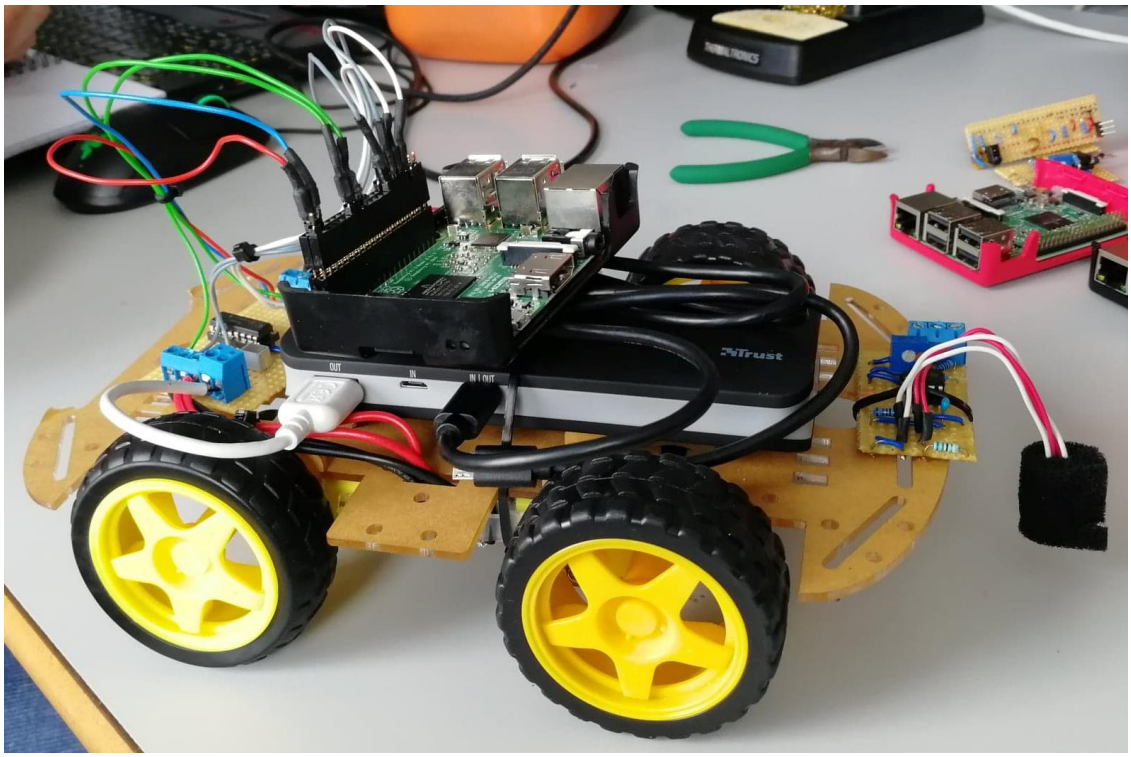
The next picture shows what the oscilloscope shows when a working line sensor board is tested.



[11] https://en.wikipedia.org/wiki/IP_address
[12] https://en.wikipedia.org/wiki/MQTT
[13] https://en.wikipedia.org/wiki/Transmission_Control_Protocol
[14] https://en.wikipedia.org/wiki/User_Datagram_Protocol
[15] https://eal-it-technology.github.io/RaspberryFun/hardware/sensor/

dain0084@edu.eal.dk
stef1571@edu.eal.dk

The last picture shows the work that has been done on the robot car base regarding mounting, the PI, motor driver, sensors and so on.



Links:

How to use Force Sensing Resistors - YouTube
LM193/LM293/LM393/LM2903 Low Power Low Offset Voltage Dual Comparators datasheet (Rev. G) - lm2903-n.pdf
tcrt5000.pdf

Date: May 15, 2019

Today we continued the work on the robot car. We mounted components on the top of the robot. We mounted the motor driver and all motors and tested again if everything worked as intended and can confirm that it is working as we want. We also mounted the line sensors to the robot base and tested again to see if everything works as intended and we can confirm one line sensor is working perfect and the other doesn't. This problem will be investigated very soon.
Lastly, we have been starting on developing the code for the robot car and its sensors and motors.

Date: May 16-17, 2019

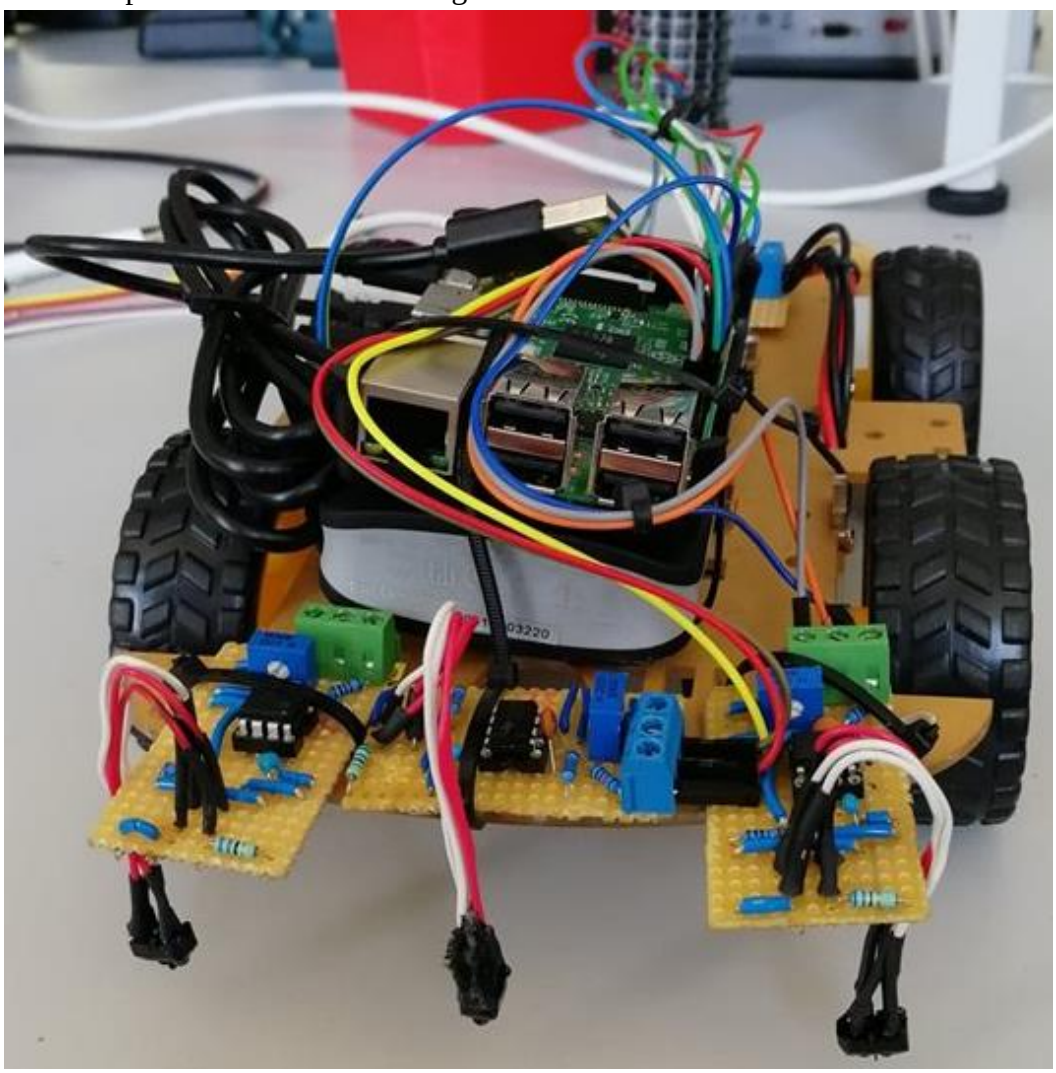
The last two days we worked on the final project report. We have been writing a lot of theory parts, proof of concept part, introduction part and other parts. We finished writing about all stuff that were possible to write about. Furthermore, we've sent a draft to our project supervisor Ilias Esmati.

Date: May 20-23, 2019

This week we finished up the robot car vehicle. We made the code, build the car, made one more line sensor board because we found out one more line sensor had problems and lastly, we tested the car with the code. The test turned out to be successful, because all motors, sensor boards and the code for the car is working now as we want it too. Last thing we must do regarding the robot car is to make it look more tidy and good looking with some cable management and better placements of the boards on the robot car base.

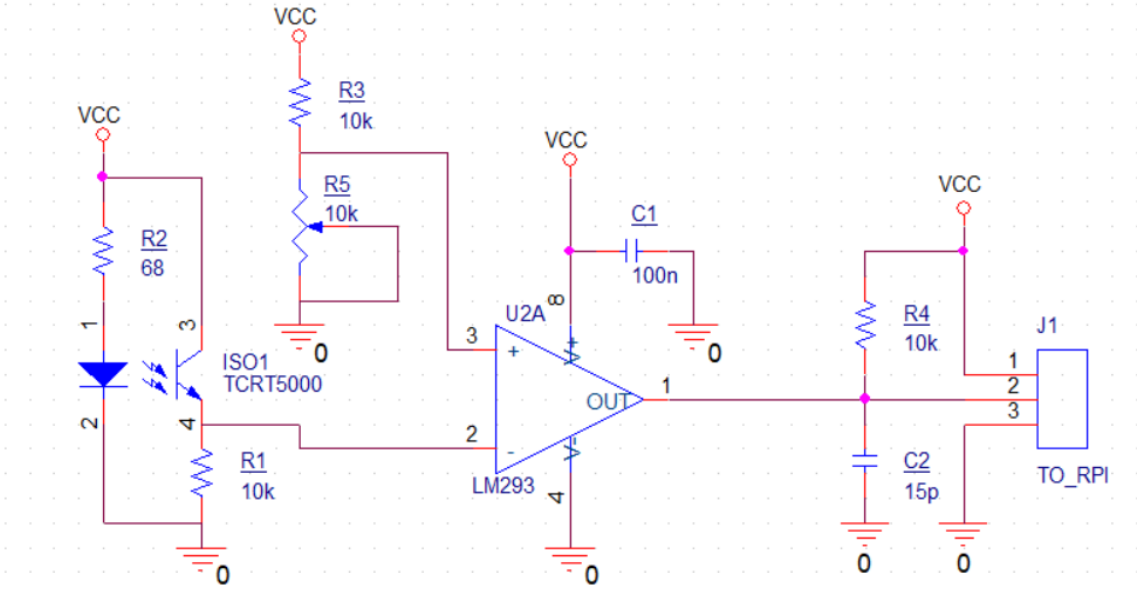
We also implemented a manual driving mode to the robot car, so if we press the keys W, A, S or D, the robot car will move.
Another thing we started on was to implement a PID controller to the robot car, so it will be running more smoothly.
Furthermore, we made a schematic for the line sensors and motor driver in the OrCAD software.
The first picture shows the testing of the new line sensor board on the oscilloscope.

The next picture shows our working robot car.

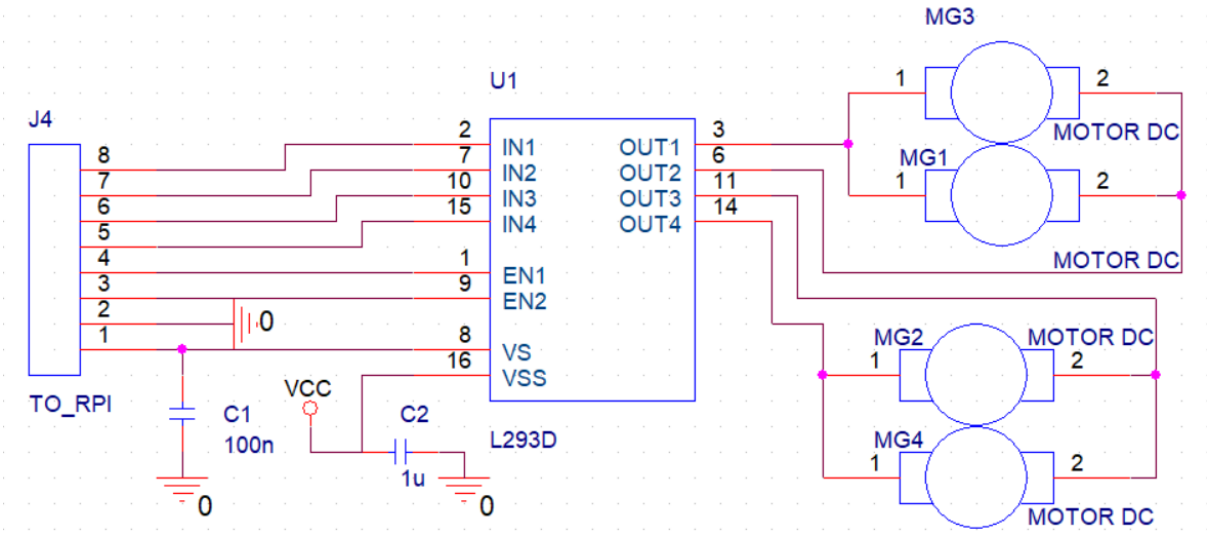


The second last picture shows the schematic we made for the line sensor boards in OrCAD.

dain0084@edu.eal.dk
stef1571@edu.eal.dk



The last picture shows the schematic we made for motor driver board in OrCAD.



Date: May 24, 2019

Today we worked on the project report the complete day and we can confirm that we have been writing about everything possible for now.

Date: May 27, 2019

Today we reordered the transmitter coils and receiver coils from a Danish store because those we ordered from Holland haven't arrived, and we are suspecting Danish customs of having the package in their custody, checking it which is the reason for it hasn't arrived.

Today we also finished our PID controlled robot car, so it works very smooth now.

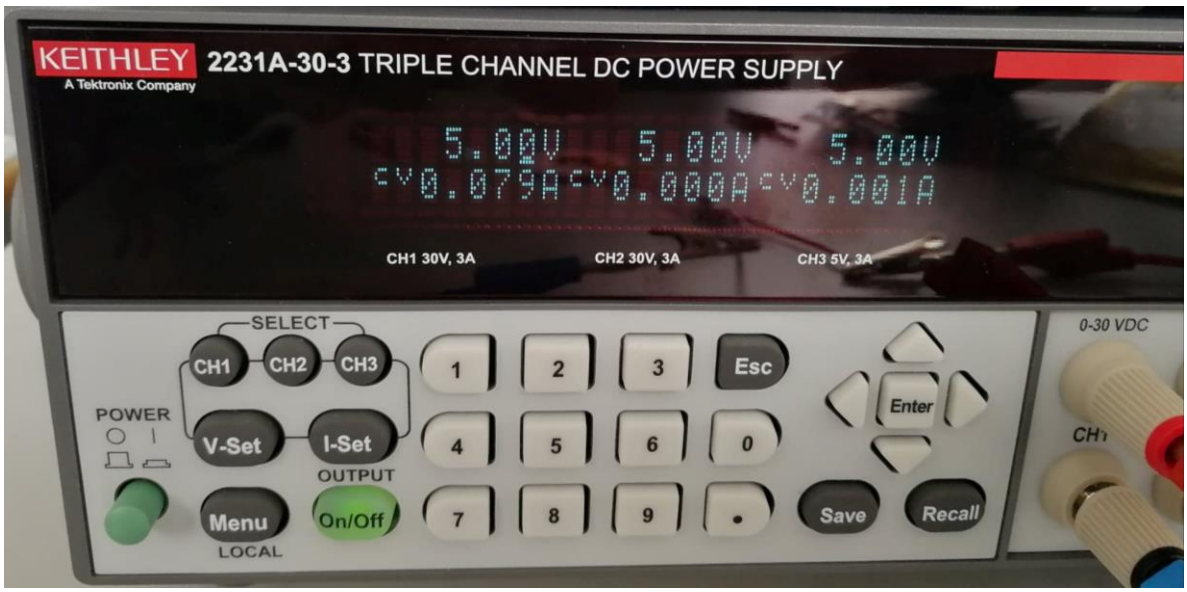
Furthermore, we finished and tested the complete TCP routine between the robot car and the charging station.

Date: May 28, 2019

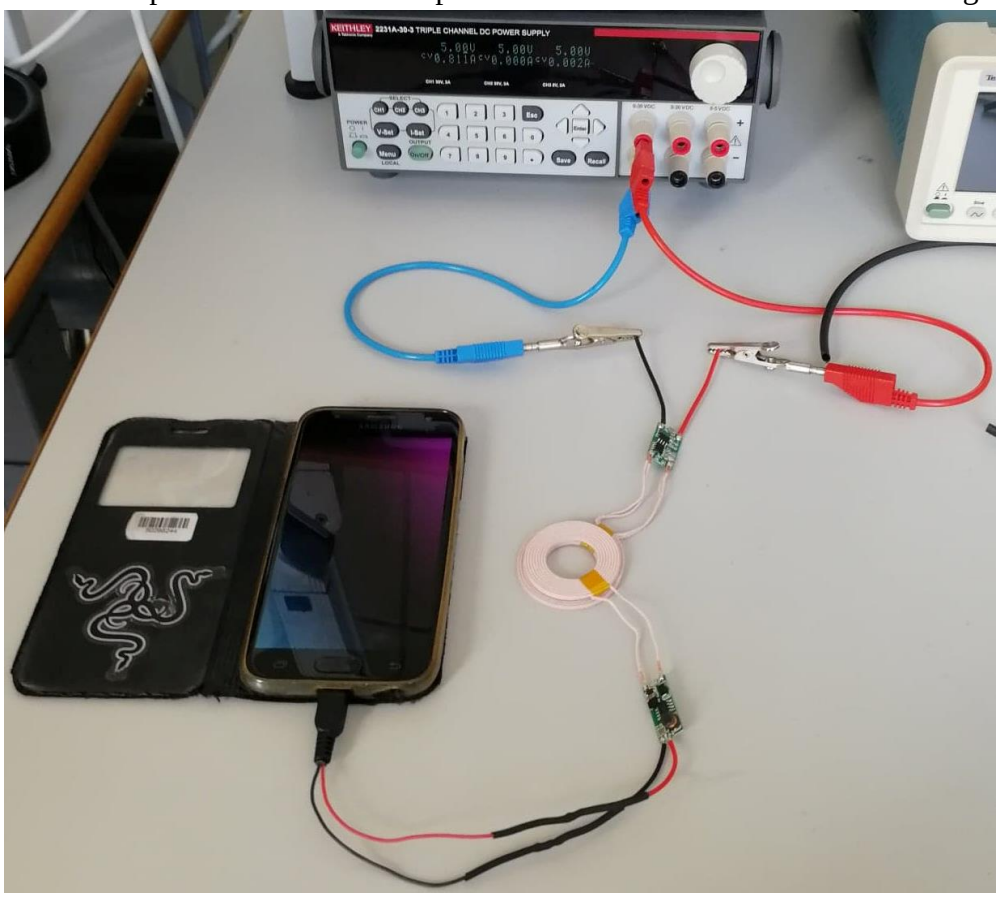
dain0084@edu.eal.dk
stef1571@edu.eal.dk

Yesterday the components/transmitter & receiver coils arrived from Holland and as we suspected, it was delayed by the Danish customs because it was being checked by them. This is a usual process for letters and packs arriving from Holland, because they have mild drug laws in their country. Furthermore, we received another delivery with two force sensitive resistors and two Raspberry Pi cameras. Now we have received all the components for our final project.
We finished up the car script completely today!
The picture below shows the reason the package was late.

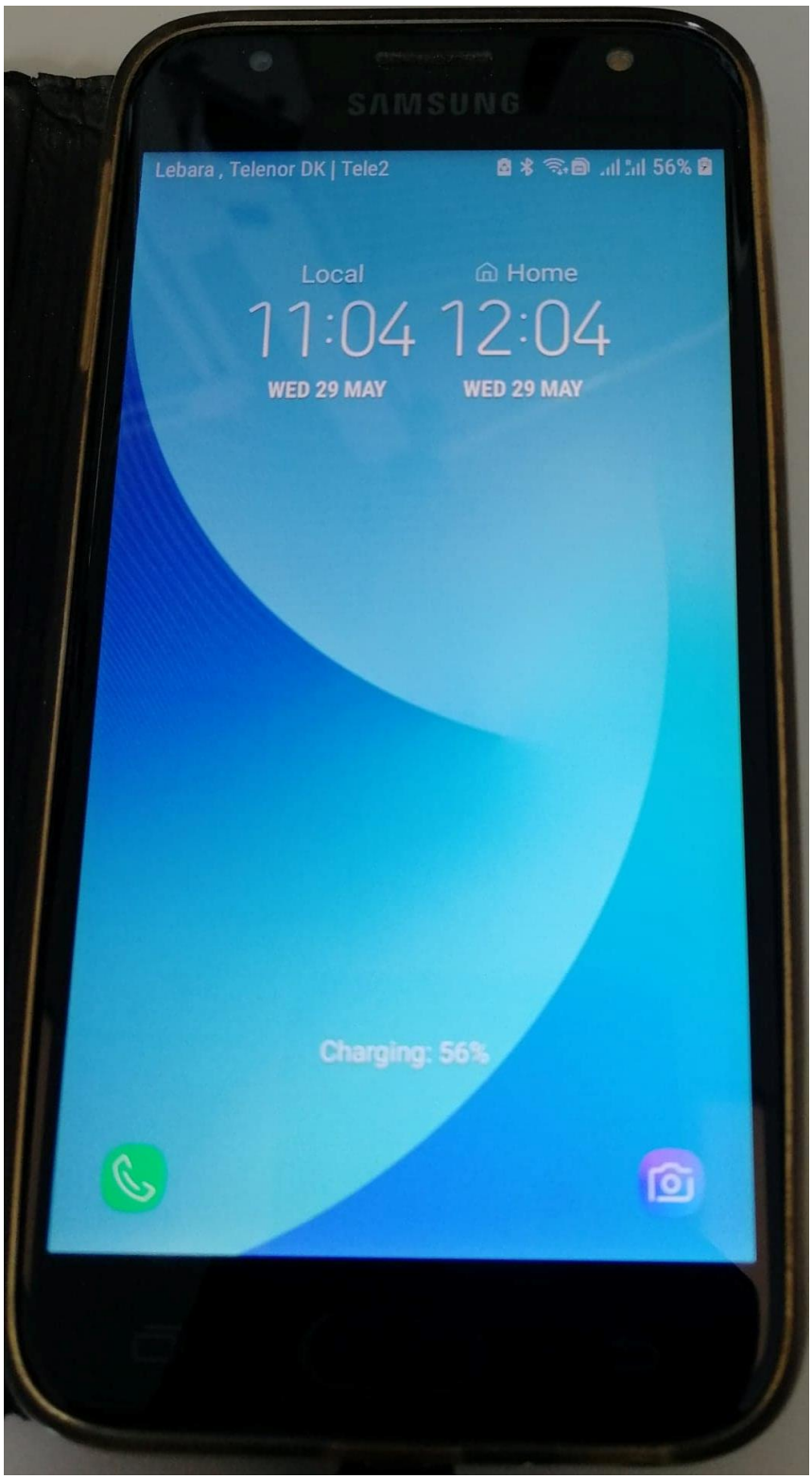
Date: May 29, 2019
We tested the wireless charging modules, connected them without a load to measure the current, and after we connected a load and measured the current.
We also started making MOSFET switches for our system, because we want to be able to trigger the coils on and off with the Raspberry Pi GPIO pins. Another thing we started making was LED boards, this is for the charging stations, this LED board can tell whether the charging station is in use or not.
Next thing we made was the UML diagrams for our software in the system and lastly, we have been writing report.
The first picture shows the testing of the coil without a load.



In the next picture we use a smartphone as a load and uses the coils to charge it wirelessly.



The last picture shows that the phone is being charged wirelessly.

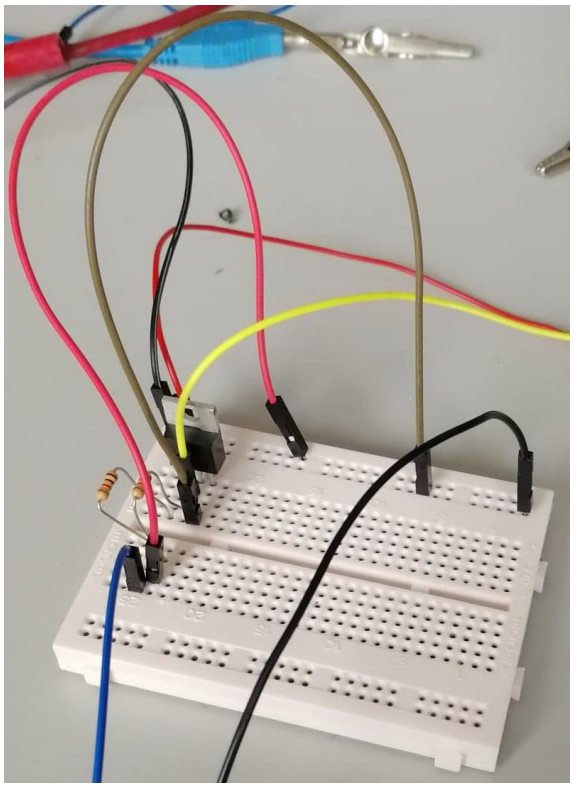


Date: May 30, 2019

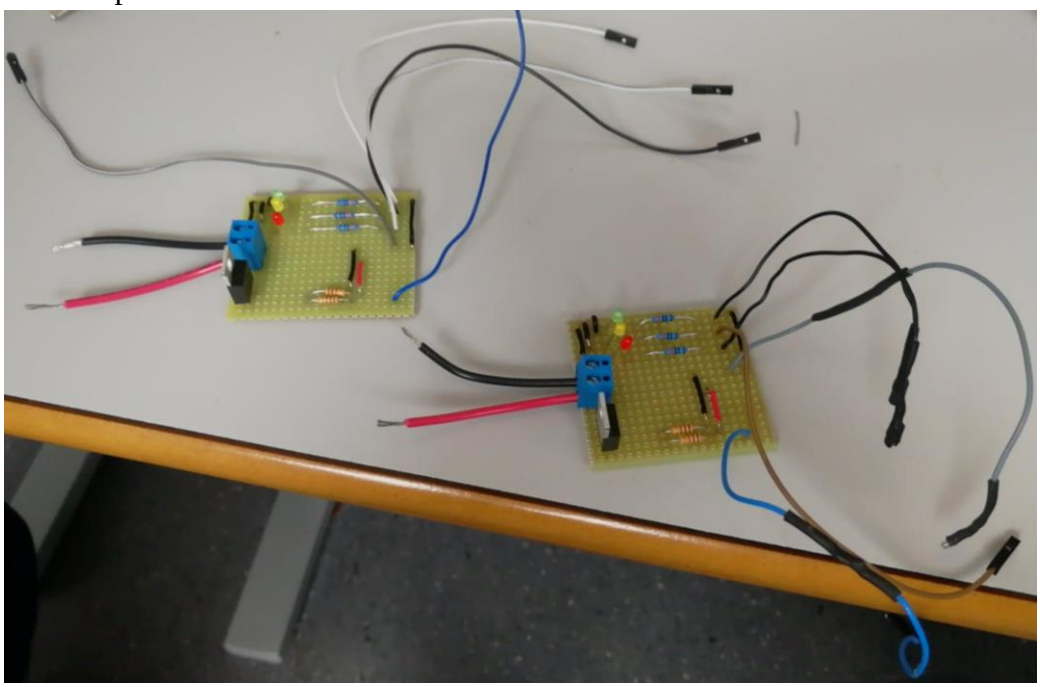
Today we finished up the MOSFET-switches and LED boards for our system. When testing the complete system with the MOSFET and LED board, we found out that the charging station didn't get enough power from our power supply to charge the robot vehicle. This can be fixed with a higher amp rated power supply. This will be bettered in the period between report deadline and the oral exam.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

Another thing we did was to connect the force sensitive resistors to our charging stations, which helps the robot car determine whether a charging station is in use or not.
Lastly, we have also been working a lot on the report.
The first picture shows the MOSFET switch circuit on a breadboard.



The last picture shows the finished MOSFET switches and LED boards on veroboard.



Date: May 31, 2019
Today and the rest of time until deadline is allocated for the project report.

dain0084@edu.eal.dk
stef1571@edu.eal.dk

We are planning on bettering the project in the time between project report hand-in and the oral exam. Things that will bettered are the power supply to our system, maybe we will also do some 3D-printed enclosures for the coils and lastly, we will maybe make a current measuring system for our project.

2. Car_script.py

Script used for the robot vehicle. Due to formatting some code may be moved to new lines.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, argparse, sys, termios, socket, re

parser = argparse.ArgumentParser()
parser.add_argument('command', type=str, default='testdrive', help='The
function to execute')
parser.add_argument('-pwm', type=int, default=50, help='Motor PWM
frequency, in Hz')
parser.add_argument('-Ku', type=float, default=400, help='Ultimate gain
for stable oscillations')
parser.add_argument('-Tu', type=float, default=425, help='Period for
stable oscillations')
parser.add_argument('-offset', type=float, default=0.1, help='Offset for
the PID controller')
parser.add_argument('-speed', type=int, default=100, help='Max duty cycle
for the motors')
parser.add_argument('-data', type=str, default="Hello, world!",
help='Data to send to station')
parser.add_argument('-TCP_IP', type=str, nargs='+', help='The IP address
array to bind to for TCP')
parser.add_argument('-TCP_PORT', type=int, default=5005, help='The socket
to bind to for TCP')
args = parser.parse_args()

def changeMotorDC(LeftDC, RightDC):
    '''
    Sets the duty cycles of the motors according to retrieved args.
    If the arg exceeds 100 or -100, the duty cycle is reset to 100.
    '''
    try:
        if(LeftDC > 0):
            GPIO.setup(LMC_1, GPIO.HIGH)
            GPIO.setup(LMC_2, GPIO.LOW)
            if(LeftDC > 100):
                LM_PWM.start(100)
            else:
                LM_PWM.start(LeftDC)
        else:
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
            GPIO.setup(LMC_1, GPIO.LOW)
            GPIO.setup(LMC_2, GPIO.HIGH)
            if(LeftDC < -100):
                LM_PWM.start(100)
            else:
                LM_PWM.ChangeDutyCycle(LeftDC * (-1))
        if(RightDC > 0):
            GPIO.setup(RMC_1, GPIO.HIGH)
            GPIO.setup(RMC_2, GPIO.LOW)
            if(RightDC > 100):
                RM_PWM.start(100)
            else:
                RM_PWM.ChangeDutyCycle(RightDC)
        else:
            GPIO.setup(RMC_1, GPIO.LOW)
            GPIO.setup(RMC_2, GPIO.HIGH)
            if(RightDC < -100):
                RM_PWM.start(100)
            else:
                RM_PWM.start(RightDC *(-1))
    except KeyboardInterrupt:
        return ("KeyboardInterrupt triggered")

def stopMotors():
    '''Stops PWM of all motors'''
    changeMotorDC(0, 0)
    LM_PWM.stop()
    RM_PWM.stop()

def testdrive():
    '''Simple def to test motor control'''
    try:
        print("Going forward in 8 secs")
        time.sleep(8)
        changeMotorDC(99, 99)
        print("Going left in 1 sec")
        time.sleep(1)
        changeMotorDC(-99, 50)
        print("Going right in 1 sec")
        time.sleep(1)
        changeMotorDC(50, -99)
        print("Going backwards in 1 sec")
        time.sleep(1)
        changeMotorDC(-99, -99)
        print("Stopping in 1 sec")
        time.sleep(1)
        stopMotors()
        return True
    except KeyboardInterrupt:
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
        return ("KeyboardInterrupt triggered")

def CheckOptoSensors():
    '''
    Creates a list that gathers the TCRT5000 sensor outputs and stores
them as strings.
    Returns output of left, center and right line sensor respectively.
    '''
    sensor = [GPIO.input(LSData), GPIO.input(CSData), GPIO.input(RSData)]
    value = str(sensor[0]) + str(sensor[1]) + str(sensor[2])
    return value

def GetPIDError():
    '''
    Sets the error for the PID controller algorithm,
    based on the sensor value array.
    '''
    value = CheckOptoSensors()
    if(value == '000'):
        error = 0
    elif(value == '001'):
        error = 15
    elif(value == '010'):
        error = 0
    elif(value == '011'):
        error = 5
    elif(value == '100'):
        error = -15
    elif(value == '101'):
        error = 0
    elif(value == '110'):
        error = -5
    elif(value == '111'):
        error = 0
    return error

def __gen_ch_getter(echo):
    def __fun():
        fd = sys.stdin.fileno()
        oldattr = termios.tcgetattr(fd)
        newattr = oldattr[:]
        try:
            if echo:
                lflag = ~(termios.ICANON | termios.ECHOCTL)
            else:
                lflag = ~(termios.ICANON | termios.ECHO)
            newattr[3] &= lflag
            termios.tcsetattr(fd, termios.TCSADRAIN, newattr)
            ch = sys.stdin.read(1)
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, oldattr)
        return ch
    return __fun
getch = __gen_ch_getter(False)
getche = __gen_ch_getter(True)

def manualdrive(speed):
    '''Enables car control using manual input'''
    print("W: accelerate, A/D: steer, S: reverse")
    print("Press C or Ctrl-C to exit")
    try:
        while (True):
            char = getch()
            if(char == "w"):
                changeMotorDC(speed, speed)
            if(char == "s"):
                changeMotorDC(-speed, -speed)
            if(char == "a"):
                changeMotorDC(-speed, speed)
            if(char == "d"):
                changeMotorDC(speed, -speed)
            if(char == "c"):
                return
            char = ""
    except KeyboardInterrupt:
        return ("KeyboardInterrupt triggered")

def PIDInit(Ku,Tu):
    '''
    Initializes the PID controller algorithm by calculating Kp, Ki and Kd
values
    from inputs Ku (ultimate gain) and Tu (oscillation period).
    By default, Ku is 400 and Tu is 425. They can be changed by
specifying them
    as additional parameters when running the script.

    The calculations are based on the Ziegler-Nichols method of PID
tuning,
    specifically the "no overshoot" settings taken from Microstar Labs:
    http://www.mstarlabs.com/control/znrule.html
    '''
    int_min = -20
    int_max = 20
    integ = 0
    deriv = 0
    lasterror = 0

    Kp = 0.2 * Ku
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
    Ki = (0.4 * Ku) / Tu
    Kd = (Ku * Tu) / 15

    print("Ku={} \nTu={} \nKp={} \nKi={} \nKd={}".format(Ku,Tu,Kp,Ki,Kd))
    return Kp,Ki,Kd,int_min,int_max,integ,deriv,lasterror

def
PIDControl(Kp,Ki,Kd,int_min,int_max,lastinteg,deriv,lasterror,speed,offse
t):
    '''
    The PID controller algorithm.
    Retrieves the error value from the sensors, and changes motor duty
cycles
    based on the error and retrieved Kp, Ki and Kd values.
    To function as intended, it needs to be executed periodically and as
fast as
    possible.
    '''
    error = GetPIDError() - offset
    integral = lastinteg + error
    if(integral > int_max):
        integral = int_max
    elif(integral < int_min):
        integral = int_min

    deriv = (error - lasterror)

    control = Kp * error + Ki * integral + Kd * deriv
    changeMotorDC(speed + control, speed - control)

    return integral, error

def PIDTest(Ku,Tu,speed,offset):
    '''
    Test def for the PID controller.
    Calls PIDInit, then periodically calls PIDControl until
KeyboardInterrupt
    is triggered.
    '''
    Kp,Ki,Kd,int_min,int_max,lastinteg,deriv,lasterror = PIDInit(Ku,Tu)
    try:
        while(True):
            integ, error =
PIDControl(Kp,Ki,Kd,int_min,int_max,lastinteg,deriv,lasterror,speed,offse
t)
            lastinteg = integ
            lasterror = error
    except KeyboardInterrupt:
        return ("KeyboardInterrupt triggered")
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
def TCP_TestSend(message,TCP_PORT,TCP_IP):
    '''
    Def for testing TCP connections with the stations.

    Input:
    message = The data you wish to send to the server
    TCP_IP = The IP of the server in the LAN
    TCP_PORT = The socket port that the server is listening to

    Output:
    First 20 bytes of message, since the server just echoes it back
    After echoed data is received, the connection is closed.
    '''
    BUFFER_SIZE = 1024
    IsConnected = False
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    for entry in range(len(TCP_IP)):
        try:
            print("Connecting to:",TCP_IP[entry])
            s.connect((TCP_IP[entry], TCP_PORT))
            IsConnected = True
            print("Connection successful")
            break
        except:
            pass
    if (IsConnected == False):
        print("No servers are responding")
        return False
    s.sendall(message.encode('utf-8'))
    data = s.recv(BUFFER_SIZE)
    s.close()
    return ("received data:", data)

def AdjustLine(speed,direction):
    try:
        value = CheckOptoSensors()
        if(value == '111'):
            pass
        else: return None
        if (direction == "left"):
            while(value != '001'):
                changeMotorDC(-speed, speed)
                time.sleep(0.01)
                value = CheckOptoSensors()
            stopMotors()
            return True
        elif (direction == "right"):
            while(value != '100'):
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
                changeMotorDC(speed, -speed)
                time.sleep(0.01)
                value = CheckOptoSensors()
            stopMotors()
            return True
        else: return False
    except KeyboardInterrupt:
        return ("KeyboardInterrupt triggered")

def ChargeCar_Line(TCP_PORT,Ku,Tu,speed,offset,TCP_IP):
    '''
    1. Attempts to connect to specified TCP_IP(s) and port.
    2. Sends charge request to the stations connected.
    3. When first reply is received, checks received station ID.
    4. If ID = 1, adjust to left side, if ID = 2, adjust to right side.
Lame, we know.
    5. Enable PID controlled movement, don't stop until station tells us
to.
    6. If stop command received, stop car. We should be charging now.
    '''
    BUFFER_SIZE = 1024
    IsConnected = False
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    for entry in range(len(TCP_IP)):
        try:
            print("Connecting to:",TCP_IP[entry])
            s.connect((TCP_IP[entry], TCP_PORT))
            IsConnected = True
            print("Connection successful")
            break
        except:
            pass
    if (IsConnected == False):
        print("No servers are responding")
        return False
    data = ""
    while (True):
        s.sendall("ChargeRequest".encode('utf-8'))
        data = s.recv(BUFFER_SIZE)
        if (data != ""):
            break
    try:
        StrData = data.decode('utf-8').strip('\x00')
        print("Received:",StrData)
        StationID = StrData[StrData.find("#")+1]
    except:
        print("Invalid data:", data)
        return False
    if (StationID == '1'):
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
        print("Adjusting to the left!")
        AdjustLine(speed * 0.5,"left")
        time.sleep(0.5)
    elif (StationID == '2'):
        print("Adjusting to the right!")
        AdjustLine(speed * 0.5,"right")
        time.sleep(0.5)
    else:
        print("This ID is not supported:",StationID)
        return False
    s.setblocking(0)
    stopdata = ""
    Kp,Ki,Kd,int_min,int_max,lastinteg,deriv,lasterror = PIDInit(Ku,Tu)
    while(stopdata != b"STOP"):
        integ, error =
PIDControl(Kp,Ki,Kd,int_min,int_max,lastinteg,deriv,lasterror,speed,offse
t)
        lastinteg = integ
        lasterror = error
        try:
            stopdata = s.recv(BUFFER_SIZE)
        except:
            pass
    stopMotors()
    s.setblocking(1)
    s.close()
    print("Arrived at destination")
    return True

class CommandList:
    '''
    Python KV dictionary containing list of available commands to be
passed as additional args.
    The dictionary is used similar to switch-case in C or C++, since
Python doesn't have
    a native switch-case solution.
    It is more efficient than if..elif..elif..else, and supports addition
of new commands easily,
    even at run-time.
    '''
    def argtodef(self, argument):
        cmdlist = {
            'testdrive': testdrive,
            'testsense': CheckOptoSensors,
            'adjustline': lambda: AdjustLine(args.speed, "right"),
            'manual': lambda: manualdrive(args.speed),
            'testtcp': lambda:
TCP_TestSend(args.data,args.TCP_PORT,args.TCP_IP),
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
            'testpid': lambda:
PIDTest(args.Ku,args.Tu,args.speed,args.offset),
            'line': lambda:
ChargeCar_Line(args.TCP_PORT,args.Ku,args.Tu,args.speed,args.offset,args.
TCP_IP),
        }
        func = cmdlist.get(argument, lambda: print("Invalid command:
{}".format(argument)))
        result_list = func()
        return result_list

def argumentparse(command):
    '''
    This def takes the parsed args and runs them through the KV
dictionary above.
    Afterwards, cleans the GPIO and returns a result, if there is one.
    '''
    try:
        cmdlist = CommandList()
        result = cmdlist.argtodef(command)
        GPIO.cleanup()
        return result
    except KeyboardInterrupt:
        print("KeyboardInterrupt")
        GPIO.cleanup()
        return False
    except:
        raise

if (__name__ == "__main__"):
    '''
    Setting up the robot car. You can change the pin numbers according to
your own setup.
    '''
    GPIO.setmode(GPIO.BCM)  # Setting GPIO setmode to BCM
    LMC_1 = 0    # Front Left Motor Control 1
    LMC_2 = 5    # Front Left Motor Control 2
    LMEn = 27    # Front Left Motor Enable
    RMC_1 = 6    # Front Right Motor Control 1
    RMC_2 = 13   # Front Right Motor Control 2
    RMEn = 22    # Front Right Motor Enable

    LSData = 26  # Left Sensor Data input
    CSData = 12  # Center Sensor Data input
    RSData = 16  # Right Sensor Data input

    CS_VCC = 4   # VCC output for Center Sensor (left and right sensors
use dedicated 3.3V outputs)
    output_list = [LMC_1,LMC_2,LMEn,RMC_1,RMC_2,RMEn]
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
    input_list = [LSData,CSData,RSData]

    GPIO.setup(output_list, GPIO.OUT) # Setting up output pins in the Pi
    GPIO.setup(input_list, GPIO.IN)   # Setting up input pins in the Pi
    GPIO.setup(CS_VCC, GPIO.OUT, initial=GPIO.HIGH)    # Since the Pi
only has 2 dedicated 3.3V outputs, we add an additional 3.3V supply from
a GPIO pin

    LM_PWM = GPIO.PWM(LMEn, args.pwm)
    RM_PWM = GPIO.PWM(RMEn, args.pwm)
    LM_PWM.start(0)
    RM_PWM.start(0)

    result = argumentparse(args.command)
    print (result)
```

3. Station_script.py

Script used for the charging stations. Due to formatting some code may be moved to new lines.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time, argparse, socket

parser = argparse.ArgumentParser()
parser.add_argument('command', type=str, default='testtcp', help='The
function to execute')
parser.add_argument('-TCP_PORT', type=int, default=5005, help='The socket
to bind to for TCP')
parser.add_argument('-StationID', type=int, default=1, help='The ID of
the station passed to car')

args = parser.parse_args()

def TCP_Testserve(TCP_PORT):
    '''
    Def for testing TCP server setups for the stations.
    Default port for listening is 5005, but can be changed with TCP_PORT
arg.
    Must be started BEFORE attempting to connect with clients.
    Once it receives data, echoes it back to client, then closes the
socket.
    '''
    BUFFER_SIZE = 20

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(('0.0.0.0', TCP_PORT))
    print("Listening...")
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
    s.listen(1)

    conn, addr = s.accept()
    print ('Connection address:', addr)
    while (True):
        data = conn.recv(BUFFER_SIZE)
        if (not data): break
        print ("received data from client: ", data)
        conn.send(data)
    conn.close()
    return True

def DisableLED():
    try:
        GPIO.output(LED_Red, GPIO.LOW)
        GPIO.output(LED_Yellow, GPIO.LOW)
        GPIO.output(LED_Green, GPIO.LOW)
        return True
    except:
        raise

def EnableLED(color):
    try:
        DisableLED()
        GPIO.output(color, GPIO.HIGH)
        return True
    except:
        raise

def TestLED():
    try:
        while(True):
            EnableLED(LED_Red)
            time.sleep(1)
            EnableLED(LED_Yellow)
            time.sleep(1)
            EnableLED(LED_Green)
            time.sleep(1)
    except KeyboardInterrupt:
        print("KeyboardInterrupt triggered")
        return None
    except:
        raise

def ToggleCoil(state):
    if (state):
        GPIO.output(MosfetPin, GPIO.LOW)
        return True
    else:
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
        GPIO.output(MosfetPin, GPIO.HIGH)
        return True

def TestCoil():
    try:
        while(True):
            print("Enabling coil for 10 secs")
            ToggleCoil(True)
            time.sleep(10)
            print("Disabling coil for 10 secs")
            ToggleCoil(False)
            time.sleep(10)
    except KeyboardInterrupt:
        print("KeyboardInterrupt triggered")
        return None
    except:
        raise

def CheckPressurePad():
    if (GPIO.input(PSInput) == GPIO.LOW):
        return True
    else:
        return False

def EnableStation(TCP_PORT,StationID):
    '''
    1. Check status by checking pressure pad value.
    if there is pressure, wait for 1sec, then try again. No precise
timing needed here.
    2. If its free, Start a TCP server and listen. Keep checking the
pressure pad.
    3. If request received, establish connection and reply w/ station ID.
    4. Check pressure pad value again.
    if no pressure, try again. The more frequent the checks the better.
    5. If there is pressure, send TCP stop command to car, then change
status.
    6. Loop back to step 1.
    '''
    print("My ID is :",StationID)
    try:
        s = None
        while (True):
            EnableLED(LED_Red)
            while(not CheckPressurePad()):
                time.sleep(0.1)
            BUFFER_SIZE = 20
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            s.bind(('0.0.0.0', TCP_PORT))
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
            s.listen(1)
            EnableLED(LED_Green)
            ToggleCoil(False)
            print("Station is free. Listening for hungry robot cars...")
            IsConnected = False
            while (CheckPressurePad()):
                try:
                    s.setblocking(0)
                    conn, addr = s.accept()
                    IsConnected = True
                    break
                except:
                    pass
            if (not IsConnected):
                print("Station is no longer free. Disabling TCP
server...")
                s.setblocking(1)
                s.close()
                continue
            print ('Request from IP:', addr)
            data = conn.recv(BUFFER_SIZE)
            if (not data): break
            print ("Received message: ", data)
            if (data == b"ChargeRequest"):
                conn.send(("FREE:#{}".format(StationID)).encode('utf-8'))
                print("Response sent")
                EnableLED(LED_Yellow)
            else:
                conn.send("InvalidCmd".encode('utf-8'))
                conn.close()
                continue
            while (True):
                if (not CheckPressurePad()):
                    print("The car has arrived!")
                    break
            conn.send("STOP".encode('utf-8'))
            ToggleCoil(True)
            conn.close()
            s.close()
            print("Charging...")
    except OSError:
        print("OSError: Address already in use")
    except KeyboardInterrupt:
        print("KeyboardInterrupt triggered")
        if (s != None):
            s.close()
        return None
    except:
        raise
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
class CommandList:
    '''
    Python KV dictionary containing list of available commands to be
passed as additional args.
    The dictionary is used similar to switch-case in C or C++, since
Python doesn't have
    a native switch-case solution.
    It is more efficient than if..elif..elif..else, and supports addition
of new commands easily,
    even at run-time.
    '''
    def argtodef(self, argument):
        cmdlist = {
            'testpressure': lambda: CheckPressurePad(),
            'testled': lambda: TestLED(),
            'testcoil': lambda: TestCoil(),
            'testtcp': lambda: TCP_Testserve(args.TCP_PORT),
            'enable': lambda:
EnableStation(args.TCP_PORT,args.StationID),
        }
        func = cmdlist.get(argument, lambda: print("Invalid command:
{}".format(argument)))
        result_list = func()
        return result_list

def argumentparse(command):
    '''
    This def takes the parsed args and runs them through the KV
dictionary above.
    Afterwards, cleans the GPIO and returns a result, if there is one.
    '''
    try:
        cmdlist = CommandList()
        result = cmdlist.argtodef(command)
        GPIO.cleanup()
        return result
    except KeyboardInterrupt:
        print("KeyboardInterrupt")
        GPIO.cleanup()
        return False
    except:
        raise

if (__name__ == "__main__"):
    '''
    Setting up the station. You can change the pin numbers according to
your own setup.
    '''
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk

```
GPIO.setmode(GPIO.BCM)  # Setting GPIO setmode to BCM
LED_Red = 5          # Red LED
LED_Yellow = 6    # Yellow LED
LED_Green = 13    # Green LED
MosfetPin = 12      # MOSFET control pin for coil toggle
PSInput = 26 # Pressure Sensor input

output_list = [LED_Red,LED_Yellow,LED_Green,MosfetPin]

GPIO.setup(output_list, GPIO.OUT) # Setting up output pins in the Pi
GPIO.output(MosfetPin, True)
GPIO.setup(PSInput, GPIO.IN) # Setting up input pins in the Pi

result = argumentparse(args.command)
print (result)
```


dain0084@edu.eal.dk
stef1571@edu.eal.dk