

VŠB – Technical University of Ostrava

University study programmes

**Autonomous navigation of the volcano observation robot**  
**Autonomní navigace robota pro pozorování sopek**

Student:

Vladan Najdek

Supervisor:

doc. Ing. Marek Babiuch, Ph.D.

Year:

2018

# Diploma Thesis Assignment

Student: **Bc. Vladan Najdek**  
Study Programme: N3943 Mechatronics  
Study Branch: 3906T006 Mechatronic Systems  
Title: **Autonomous Navigation of the Volcano Observation Robot**  
**Autonomní navigace robota pro pozorování sopek**  
The thesis language: English

## Description:

1. Familiarize yourself with the volcano observation robot CLOVER.
2. Implement a motion control system based on a Raspberry Pi and ROS frameworks.
3. Utilizing ROS frameworks, get the view from a camera into the operator's laptop.
4. Discuss possible means of an autonomous navigation of the robot into the transportation net and choose the optimal one.
5. Develop electronic circuits for the chosen navigation procedure.
6. Create software that implements the chosen navigation procedure.
7. Evaluate the achieved result.

## References:

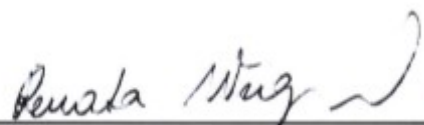
- ALCHIN, Marty. Pro Python. Berkeley: Apress, c2010. ISBN 978-1-4302-2757-1.  
BERGER, Arnold. Embedded systems design: an introduction to processes, tools, and techniques. Lawrence: CMP Books, c2002. ISBN 1-57820-073-3.  
BRÄUNL, Thomas. Embedded robotics: mobile robot design and applications with embedded systems. Berlin: Springer, c2003. ISBN 3-540-03436-6.  
PILGRIM, Mark. Ponofme se do Python(u) 3: Dive into Python 3. Praha: CZ.NIC, c2010. ISBN 978-80-904248-2-1.  
ROS Installation: ROS Installation Options [online]. Open Source Robotic Foundation, 2017 [cit. 2017-11-30]. Dostupné z: <http://wiki.ros.org/ROS/Installation>  
SIEGWART, Roland, Illah Reza NOURBAKSH a Davide SCARAMUZZA. Introduction to autonomous mobile robots. 2nd ed. Cambridge: MIT Press, c2011. ISBN 978-0-262-01535-6.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **doc. Ing. Marek Babiuch, Ph.D.**

Date of issue: 08.12.2017

Date of submission: 21.05.2018

  
doc. Ing. Renata Wagnerová, Ph.D.  
Head of Department



  
Ing. Zdeňka Chmelíková, Ph.D.  
Vice-rectress for Study Affairs

### **Prohlášení studenta**

Prohlašuji, že jsem celou diplomovou práci včetně příloh vypracoval samostatně pod vedením vedoucího diplomové práce a uvedl jsem všechny použité podklady a literaturu.

V Ostravě: .....

.....

podpis studenta

Prohlašuji, že:

- jsem byl seznámen s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo.
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen „VŠB-TUO“) má právo nevýdělečně ke své vnitřní potřebě diplomovou práci užít (§ 35 odst. 3).
- souhlasím s tím, že diplomová práce bude v elektronické podobě uložena v Ústřední knihovně VŠB-TUO k nahlédnutí a jeden výtisk bude uložen u vedoucího diplomové (bakalářské) práce. Souhlasím s tím, že údaje o kvalifikační práci budou zveřejněny v informačním systému VŠB-TUO.
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona.
- bylo sjednáno, že užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).
- beru na vědomí, že odevzdáním své práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, bez ohledu na výsledek její obhajoby.

V Ostravě dne: .....

.....

podpis studenta

Jméno a příjmení autora práce:

Vladan Najdek

Adresa trvalého pobytu autora práce:

Matušínského 1044/8 716 00 Ostrava – Radvanice

## **Acknowledgement**

I would like to thank assoc. professor Keiji Nagatani for allowing me to stay in his laboratory and conduct this research.

I would also like to thank doc. Ing. Marek Babiuch, Ph.D. for the help with finishing this thesis.

## **Annotation**

NAJDEK, V. *Autonomous navigation of the volcano observation robot*. Ostrava: VŠB – Technical University of Ostrava, University Study Programmes, 2018, 60 p. Thesis head: doc. Ing. Marek Babiuch, Ph.D.

The purpose of this thesis was to design a method of autonomous navigation to the transportation net of the volcano observation robot CLOVER. In the beginning, it describes how the robot operates, then it is described how two old robots were merged together to recreate the CLOVER robot based on Raspberry Pi a ROS frameworks. Next, various methods of autonomous navigation are discussed to determine which one is optimal for the CLOVER robot in terms of reliability, weight and price of the required hardware. The chosen method uses infrared beacon suspended from the top of the transportation net and infrared receiver mounted on the robot. Finally, there is a description of the implementation of the navigation procedure.

## **Keywords**

Autonomous navigation, robot, Raspberry Pi, ROS, infrared sensor

## **Anotace**

NAJDEK, V. *Autonomní navigace robota pro pozorování sopek*. Ostrava: VŠB – Technická univerzita Ostrava, Univerzitní studijní programy, 2018, 60 s. Vedoucí práce: doc. Ing. Marek Babiuch, Ph.D.

Cílem této práce bylo navrhnout metodu autonomní navigace robota pro pozorování sopek CLOVER do jeho transportační sítě. V práci je nejprve popsáno, jak robot pracuje, a poté jak byl ze dvou starých robotů vytvořen robot CLOVER, založen na Raspberry Pi a frameworku ROS. Dále následuje rozbor metod autonomní navigace s následným výběrem optimální metody z hlediska spolehlivosti, váhy a ceny potřebných zařízení. Vybraná metoda využívá infračervený vysílač zavěšený na vrcholku transportační sítě a infračervený přijímač nasazený na robota. Nakonec je popsána implementace navrženého řešení.

## **Klíčová slova**

Autonomní navigace, robot, Raspberry Pi, ROS, infračervený senzor

## Contents

List of abbreviations .....	8
List of units .....	9
Introduction .....	10
1 Description of the robot.....	11
2 Implementation of a new control system .....	17
2.1 Software .....	18
3 Camera .....	21
4 Autonomous navigation .....	27
5 Implementation of the proposed solution.....	33
5.1 Beacon design .....	34
5.2 Receiver design.....	37
5.3 Navigation script.....	41
6 IR navigation of the Arduino robot.....	46
6.1 Infrared beacon .....	46
6.2 Infrared receiver.....	50
6.3 Navigation algorithm .....	51
Conclusion.....	58
Literature .....	60
Appendix .....	61

## **List of abbreviations**

DC – Direct Current

GPIO – General purpose input/output

IP – Internet protocol

I<sup>2</sup>C – Inter-Integrated Circuit

IR - Infrared

LAN – Local Area Network

LED – Light Emitting Diode

LiFe – Lithium Ferrite

Li-Ion – Lithium Ion

PWM – Pulse Width Modulation

ROS – Robot Operating System

SPI – Serial Peripheral Interface

SSH – Secure Shell

TV – Television

UAV – Unmanned Aerial Vehicle

UGV – Unmanned Ground Vehicle

USART - Universal Synchronous/Asynchronous Receiver/Transmitter

USB – Universal Serial Bus

VM – Virtual Machine



## List of units

° – Degree

$\Omega$  – Ohm

k $\Omega$  – Kiloohm

% - Percent

A – Ampere

Ah – Ampere-hour

cm – Centimetre

g – Gram

Kg – Kilogram

kHz – Kilohertz

ms – Millisecond

nF – Nanofarad

Nm – Newton meter

mA – Milliamperes

mNm – Millinewtonmeter

rpm – Revolutions per minute

V – Volt

## **Introduction**

Volcano eruptions are not only dangerous in the immediate vicinity of the eruption, but also in a much wider area with their side effects. The volcanic ash is dangerous for airplanes, either blocking the vision of the pilot or damaging the plane's engines. It also blocks sunlight, lowering the temperature in the area, which in extreme cases can create what is called a volcanic winter. Volcanic gases also in many cases contain sulphur dioxide and hydrogen chloride, which cause acid rains.

Since Japan is located on a boundary of tectonic plates, it has many volcanoes. As technology advances it is expected that robots will help not only in emergency situations, but also before they happen. One of the task, where robots can help, is the observation of an active volcano to find out its state. This information is then used for example for warning or planning of evacuation.

The purpose of this research is to improve the volcano observation robot by creating means of autonomous navigation in cases where it is difficult for the teleoperator to navigate it manually.

# 1 Description of the robot

The purpose of the CLOVER robot is to observe volcanoes. It is carried in a net hanging from a UAV to the vicinity of a volcano. This requires the robot to be as light as possible, because additional weight reduces flight time of the UAV. Then the UAV descends to land the net with UGV. The robot is teleoperated using 3G connection. The teleoperator then drives the robot to the top of the volcano to conduct observation using a camera mounted on the UGV. Then it is driven back and it is navigated into a net carried by the UAV, lifted and brought back. (FieldRoboticsLaboratory, 2016)



*Fig 1.1: Robot carried by the UAV (FieldRoboticsLaboratory, 2016)*

At the time of my arrival the CLOVER was robot at the exhibition in Tokyo. Therefore, I was given a task to recreate this robot from a similar robot that wasn't needed anymore. This would allow me to experiment with my own CLOVER robot without the risk of damaging the real one and I would also learn how the robot works in the process of creating. I started with the disassembly of the old robot's electronics to see if some parts can be reused.



*Fig. 1.2: Old robot.*

The main processing unit consists of circuit board with AXIS ETRAX 100LX system-on-chip. It is designed for embedded Linux applications. It has Ethernet port and utilises Teridian 78Q2120C-CGT Ethernet transceiver. It also has AXIS ARTPEC-2 video chip. (Axis, 2016)



*Fig. 1.3: Control board.*

For controlling Maxon motors there was a motor driver with ARM STM32F103 microcontroller. The communication with this device is done using either RS485, USART, SPI or I<sup>2</sup>C bus. The problem is that it doesn't have USB for connection to a control unit.



*Fig. 1.4: Motor driver.*

The observation of the volcano is conducted using Panasonic rc90kd camera. This camera is primarily made for mounting on the back of a car to transmit image during reversing. According to the website of the manufacturer this hardware is still maintained and I think it could be reused. (Panasonic, 2016)



*Fig. 1.5: Panasonic rc90kd camera.*

I decided to reuse two Maxon motors that were already connected to wheels. These brushed motors, when rotating in the opposite directions enable robot to move either forward or backward. When the motors turn in the same direction the robot turns while remaining in place.

*Tab. 1.1: Maxon motor parameters. (Maxon, 2016c)*

Parameter	Value
Nominal voltage	12 V
Nominal speed	5720 rpm
Nominal torque (max. continuous torque)	11,7 mNm
Nominal current (max. continuous current)	0,806 A
No load speed	7470 rpm
No load current	30,9 mA
Speed constant	634 rpm/V

In this application, it is not necessary for the wheels to turn thousands of times per second. Instead we need a large torque, that would allow the robot to overcome terrain obstacles of various sizes and inclinations, that it encounters while it navigates around a volcano. For this conversion, the motors are equipped with a planetary gearbox.

*Tab 1.2: Maxon gearbox parameters. (Maxon, 2016b)*

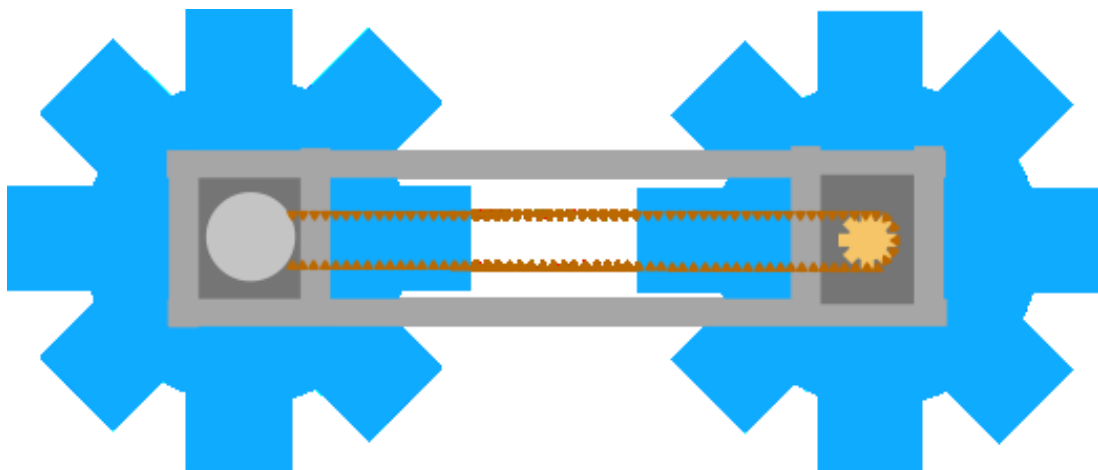
Parameter	Value
Reduction	157:1
Number of stages	3
Max. continuous torque	1,2 Nm
Max. efficiency	59 %
Weight	68 g

The motors are also equipped with differential encoders. By counting the pulses in a motor driver, we can measure the speed of rotation of the motor which allows us to gain the velocity of the robot because we know the diameter of the wheels. The encoders also allow us to get the trajectory of the robot which is helpful for the robot navigation.

*Tab. 1.3: Maxon encoder parameters. (Maxon, 2016a)*

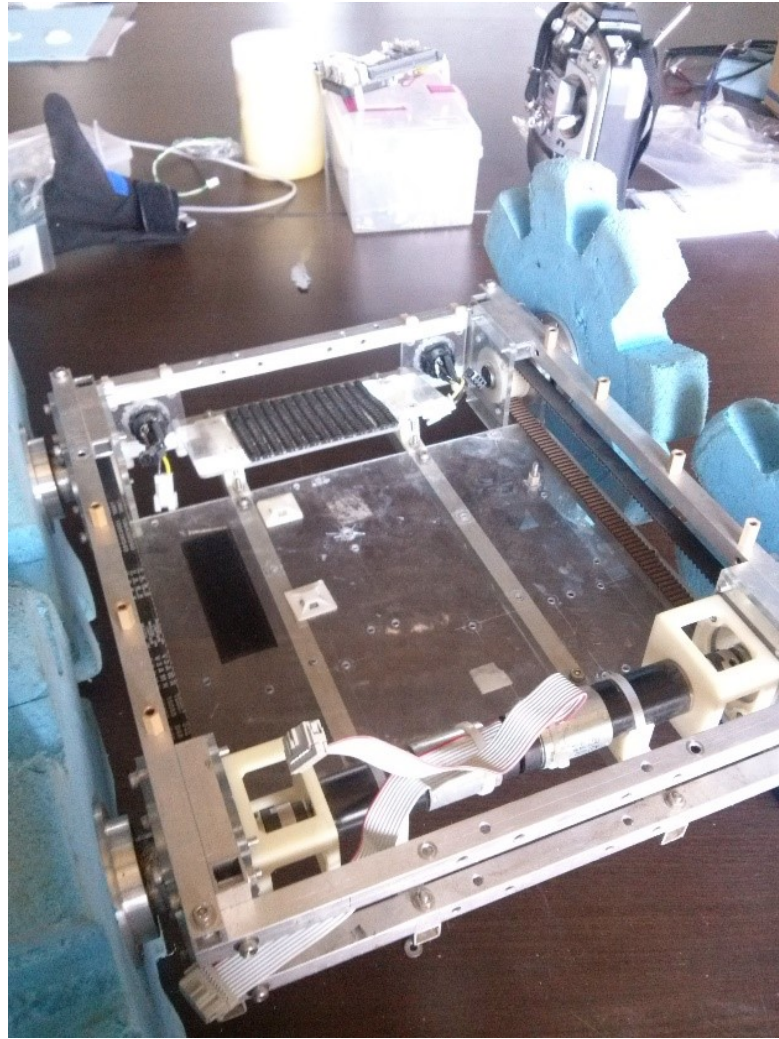
Parameter	Value
Counts per turn	128
Number of channels	2
Max. operating frequency	80 kHz
Max. speed	37 500 rpm
Supply voltage	5 V

I also decided to reuse belt transmissions, wheels and a frame made of aluminium profiles. The belts transfer torque from the front wheels to the back wheels, making the robot a four-wheel drive. This is useful because when the robot is overcoming an obstacle one of wheels might end up in the air which could result in a robot being stuck.



*Fig. 1.6: Power transmission from motor (left wheel) to back wheel.*

There were also other less important components that can be reused on other robots like a USB hub, LAN to Wi-Fi converter or different types of antennas.

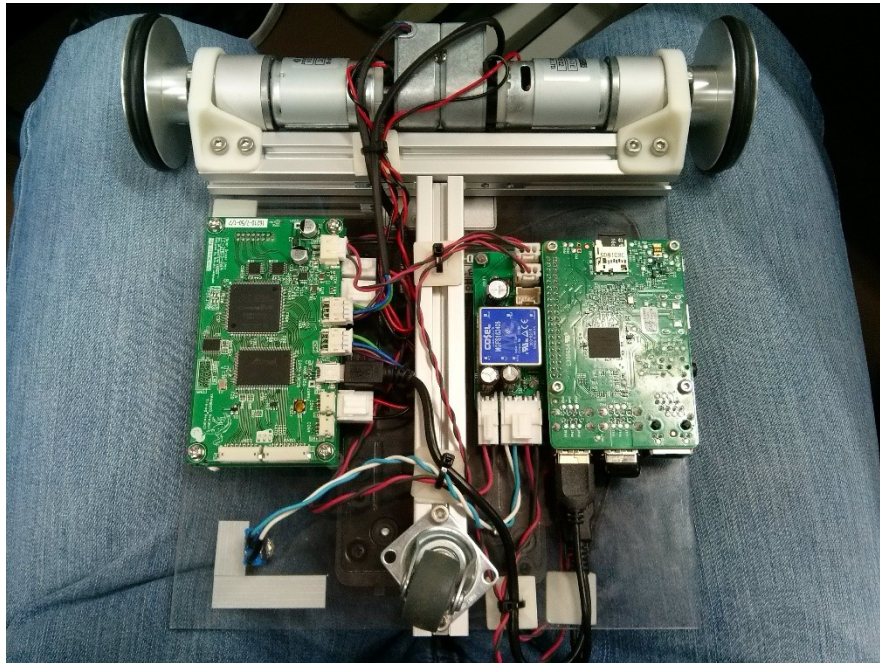


*Fig. 1.7: Disassembled robot.*



## 2 Implementation of a new control system

Since the original CLOVER robot in the field robotics laboratory uses ROS frameworks, there was a need to make this robot use ROS too. The ROS framework is developed for either Ubuntu or Debian. This means that the most suitable solution for running ROS is Raspberry Pi. I could remove a Raspberry Pi 2 from another disabled robot that was made for beginner's seminar. This Raspberry already had installed Ubuntu 14.04 and ROS indigo and contained basic scripts for communication between the microcomputer and motor driver that allowed controlling the movement of the robot using Wi-Fi. The beginner's seminar robot also has similar construction with two DC motors. (ROS wiki, 23. 5. 2016)

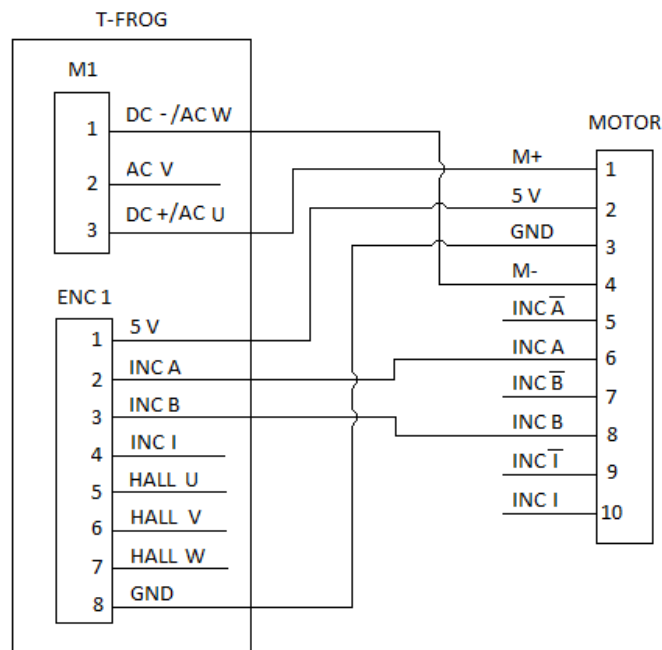


*Fig. 2.1: Beginner's seminar robot.*

Raspberry Pi on the robot cannot be powered using power outlet and USB. I decided to reuse 14.4 V, 6 Ah Li-ion battery from beginner's seminar robot which comes with a circuit board containing DC-DC converter that reduces voltage to 5 V.

Similarly to the beginner's seminar robot, I have decided to use a T-FROG motor driver. It was created specifically to use with ROS. The problem with connecting motors to the T-frog driver is that the motors of the volcano robot use a differential encoder, while motor driver has only single ended inputs. Since I am not even sure if I will need

encoders later, because the robot is teleoperated and not autonomous, I decided for now to just connect only positive signal wires.



*Fig. 2.2: Schematics of motor – driver connection.*

The communication is done over Wi-Fi. For this there is a dongle in the USB port. The Raspberry Pi has set a fixed IP of 172.16.245.10. When I needed to modify files on the microcomputer I used SSH:

```
$ssh frl@172.16.245.10
```

## 2.1 Software

The Raspberry Pi part of the software was already taken care of. I needed to prepare a computer for teleoperator. I decided to use my laptop with virtual machine running Ubuntu 14.04. First, I followed the procedure that creates ROS environment using these commands:

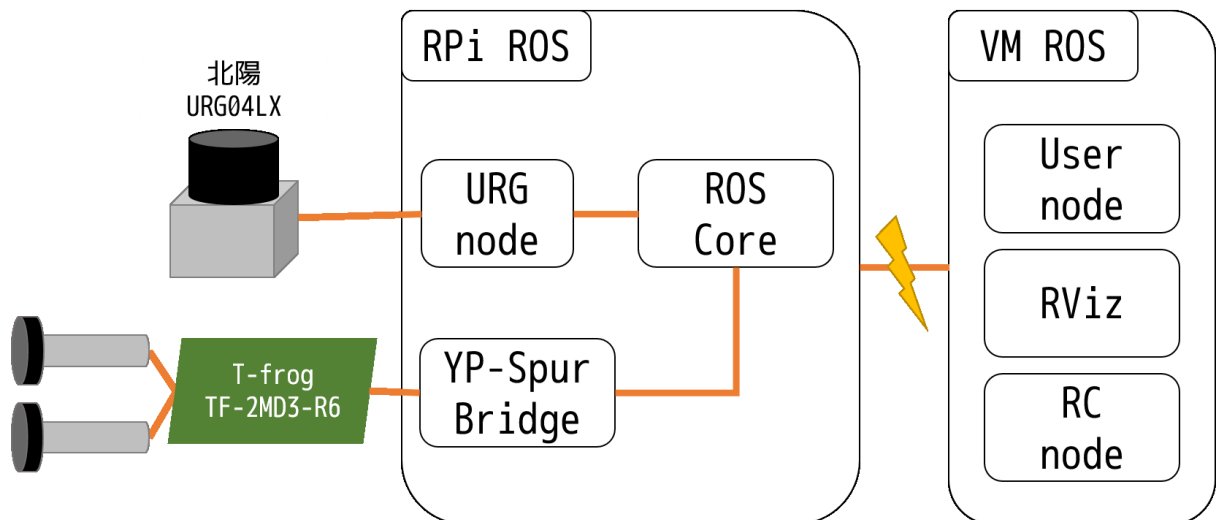
```
$mkdir -p ~/bs_ws/src
$cd ~/bs_ws/src
$catkin_init_workspace
$cd ..
$catkin_make
```

```

$source devel/setup.bash
$cd src
$git clone http://172.20.0.20:8000/ALL/BeginnersSeminar.git
$cd ..
$catkin_make

```

This creates a `bs_ws` directory for the project, then initialises the workspace and clones beginner's seminar packages `bs_mover`, `bs_rc` and `bs_start` from the gitlab of the field robotics laboratory and finally compiles them. (GitLab, 6. 2016)



*Fig. 2.3: Beginner's seminar robot control system. (GitLab, 6. 2016)*

Since the CLOVER robot doesn't have a 2D scanner I removed the URG node from the Raspberry Pi `bs_mover` launch file. Then I need to install ROS node for joystick remote control in the VM using command

```

$sudo apt-get install ros-indigo-joy

```

That's all for the installation. To control the robot, I connected PlayStation 4 joystick. In my case I found out that the joystick doesn't configure itself properly and I had to do it manually. First, I displayed all connected input devices with command

```

$ls /dev/input/

```

In my case I had three joystick devices called `js0`, `js1` and `js2`. Each of them had to be tested by running the following command, moving the analog stick and watching if there are any changes to the data displayed.

```

$sudo jstest /dev/input/jsX

```

For me it was js2 but this can change according to the number of devices connected. Now I needed to give permission to use the joystick for the ROS node with

```
$sudo chmod a+rw /dev/input/js2
```

Finally, I sent the ROS node information about which joystick device to use with

```
$rosparam set joy_node/dev "/dev/input/js2"
```

Now that I had my joypad fully configured and I was ready to connect to the robot. (ROS wiki, 12. 5. 2016)

First, I export IP address of the robot's Raspberry Pi which has its IP set to fixed

```
$export ROS_MASTER_URI=http://172.16.245.10:11311/
```

then I enter the IP address of the virtual machine which I found using `$ifconfig`

```
$export ROS_IP=IP of VM
```

It is important that both virtual machine and Raspberry Pi are on the same network. When in doubt it is enough to `$ping` the robot's IP.

Finally, I launch ROS with

```
$roslaunch bs_rc bs_rc_wo_rviz.launch
```

Another option is to use

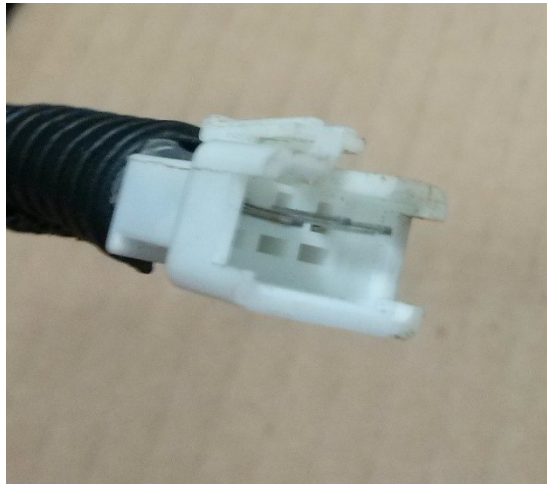
```
$roslaunch bs_rc bs_rc.launch
```

which launches visualization software called rviz for the 2D scanner of the beginner's seminar robot. This isn't necessary now but it might come in handy later for getting camera feed. (GitLab, 9. 2016)

By trying the remote control, I discovered a problem. When I pressed a joystick analog stick to the left, the robot turned right and vice versa and it was also turning very slowly. It was necessary to update the parameter file called `bs_mover.param` in a folder `catkin_ws/src/BeginnersSeminar/bs_mover/params` on Raspberry Pi.

### 3 Camera

My intent was to reuse the Panasonic rc90kd from previous version of the robot. This proved impossible because the original connector had been cut off and replaced with a different one without any documentation of rewiring.



*Fig 3.1: Rewired connector of the Panasonic camera.*

Instead I had to choose a different camera. Because my main control unit was a Raspberry Pi, it was logical to either choose an official Raspberry Pi camera module or a USB web camera. I was given an iBUFFALO web camera from a laboratory storage.

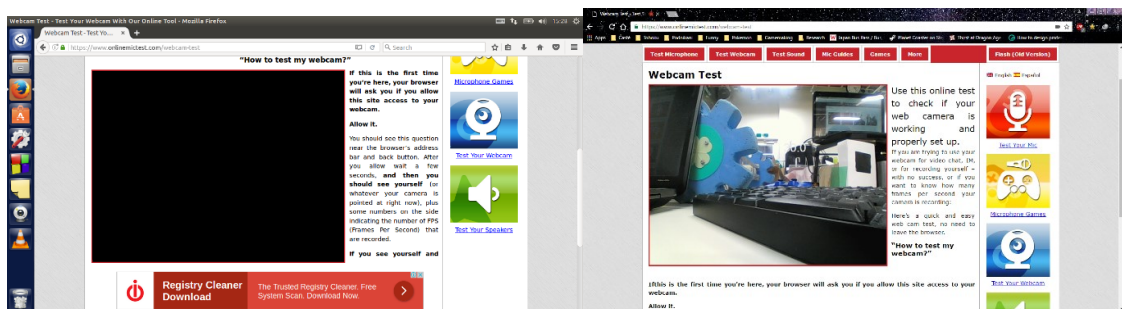


*Fig. 3.2: iBUFFALO web camera.*

First, I wanted to test whether the camera works. I connected it to the USB port and enabled it inside VirtualBox, then I went to:

<https://www.onlinemictest.com/webcam-test>

to test the camera. The result was a black screen. I decided to perform the same test on Windows with a successful result. This could mean that there were no drivers for the webcam in Ubuntu. After searching for solution, I found out that VirtualBox has issues with old USB webcams. (VirtualBox, 2014)



*Fig. 3.3: Webcam test on Ubuntu (left) and Windows (right).*

When I knew that camera is working but not with VirtualBox I wanted to test the camera with Raspberry Pi. I connected the camera to the USB port of the Raspberry Pi and used SSH to connect to it. I was afraid that there could be issues with drivers or permissions so I decided to take a simple picture with it first. For this I installed fswebcam with:

```
$sudo apt-get install fswebcam
```

Then I took a picture using:

```
$fswebcam image.jpg
```

Finally, I wanted to get the picture to my virtual machine using secure copy:

```
$scp frl@172.16.245.10:image.jpg .
```

But for some reason this didn't work. The process showed success but the file was nowhere to be found. I suspect that this was another issue of running a virtual machine. Instead of searching for a solution I decided to plug in my USB flash drive. I used

```
$sudo fdisk -l
```

to display a list of drives. I could see that my flash drive is /dev/sdb1. Then I created a new directory called /media/usb for the flash drive with

```
$sudo mkdir /media/usb
```

Finally, I mount the flash drive on the directory with

```
$sudo mount /dev/sdb1 /media/usb
```

to be able to copy the image using

```
$cp image.jpg /media/usb
```

This way I got the following image to the PC. (Askubuntu, 2012)



*Fig. 3.4: Picture taken using fswebcam.*

When I knew that the camera is working fine it was time to choose drivers that would capture the image from the camera and publish it on a ROS topic. I decided to use a `usb_cam` driver. On a Raspberry Pi I used:

```
$cd bs_ws/src
```

```
$git clone https://github.com/bosch-ros-pkg/usb_cam.git
```

To download the package into the workspace and then I used following commands to compile it:

```
$cd ..
```

```
$catkin_make
```

Then I had to make sure it doesn't get deleted by the read-only system with:

```
$rfsync-add /home/frl/bs_ws/
```

It is not necessary to source the `setup.bash` file inside the `devel` folder because this has already been done inside the `.bashrc` file. Next, I wanted to check if the dependency requirements are satisfied. For this there is a tool called `rosdep`:

```
$rosdep check
```

This showed that there were multiple dependencies missing. First, I used

```
$sudo fsprotect-chroot.sh
```

and then I installed missing packages with:

```
$sudo apt-get install ros-indigo-image-transport ros-indigo-camera-info-manager v4l-utils
```

Finally, I ran the camera node with:

```
$roslaunch usb_cam usb_cam_node.
```

On the laptop, I have to export IP addresses before I launch `rviz`:

```
$export ROS_MASTER_URI=http://172.16.245.10:11311/
```

```
$export ROS_IP=IP of VM
```

```
$roslaunch rviz rviz
```

This launches a visualization software. I clicked `Add` at the bottom left corner and chose `Image` from the `Display` type menu. This creates new window that will later be used to display camera feed. Then I need to find out on what topic does the camera node publish image. This I done using:

```
$rostopic list
```

I can see from the list that there are topics `/usb_cam/image_raw` and `/usb_cam/camera_info` that weren't there before.



```
ubuntu@ubuntu-VirtualBox: ~
ubuntu@ubuntu-VirtualBox:~$ rostopic list
/bs_mover/led1
/bs_mover/led2
/bs_mover/led3
/bs_mover/led4
/diagnostics
/hokuyo/parameter_descriptions
/hokuyo/parameter_updates
/rosout
/rosout_agg
/scan
/tf
/tf_static
/usb_cam/camera_info
/usb_cam/image_raw
/yjspur/ad/ad0
/yjspur/ad/ad1
/yjspur/ad/ad2
/yjspur/ad/ad3
/yjspur/ad/ad4
/yjspur/ad/ad5
/yjspur/ad/ad6
/yjspur/ad/ad7
/yjspur/cmd_vel
```

Fig. 3.5: List of ROS topics.

Therefore, I have to write /usb\_cam/image\_raw into the Image Topic field.

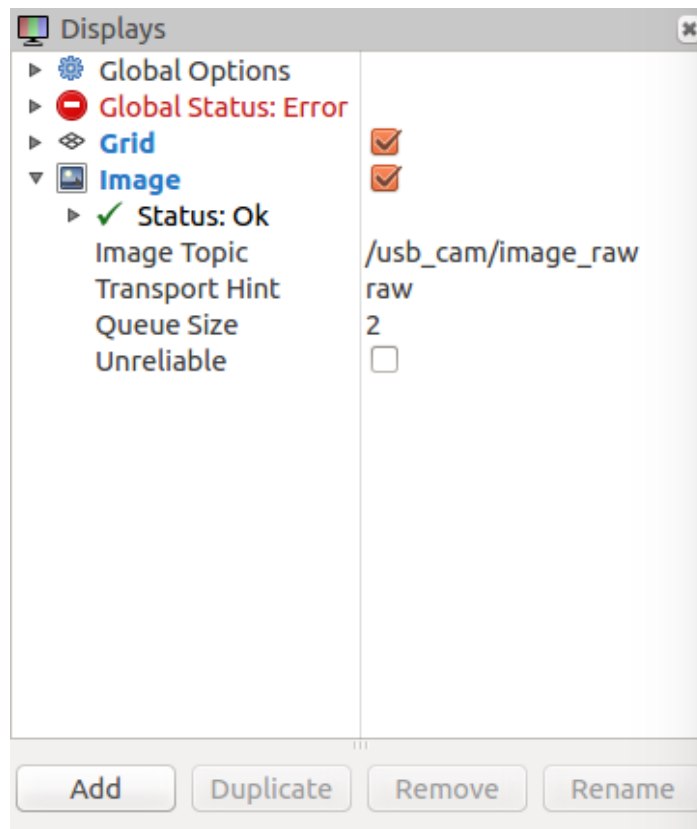
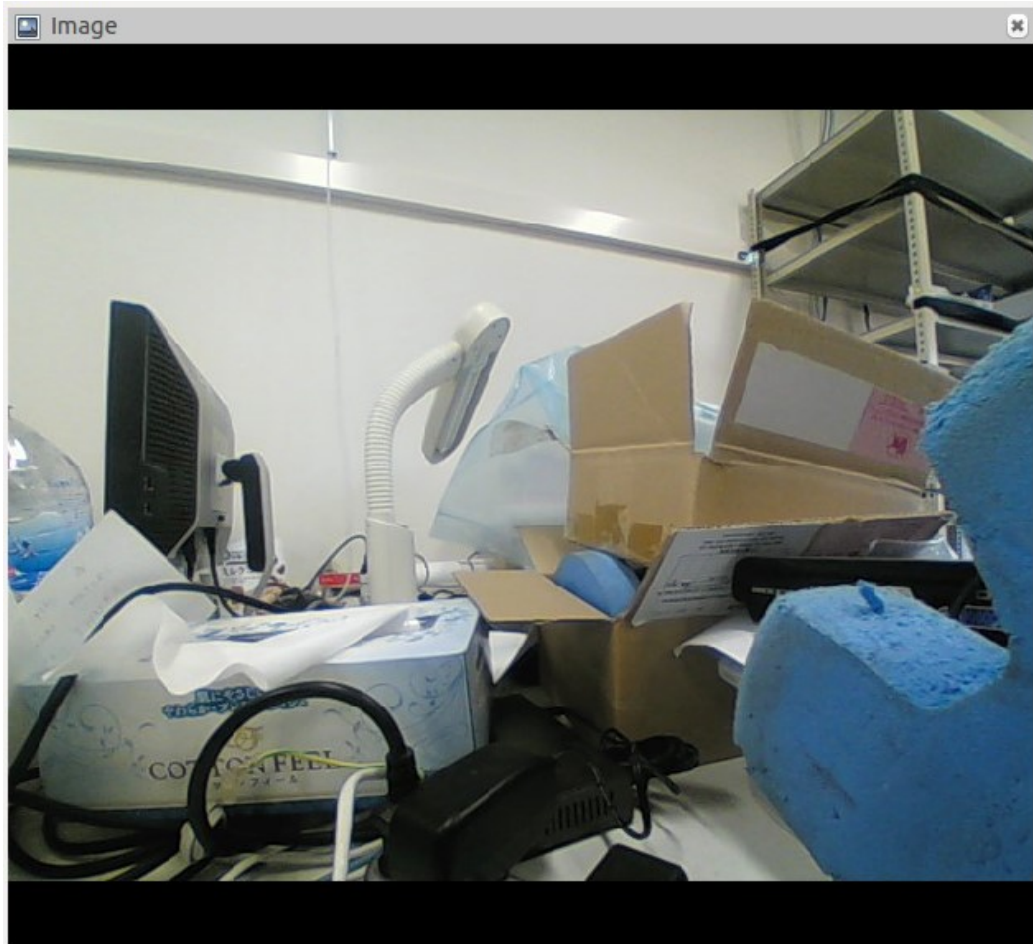


Fig. 3.6: Display tab of the rviz software.

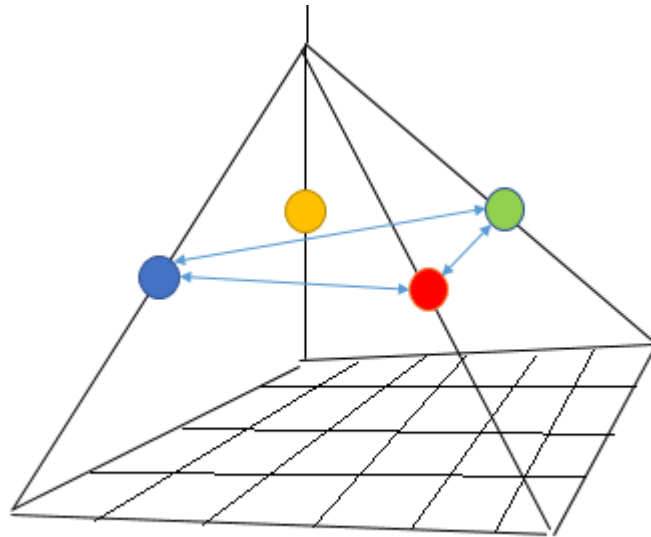
The next step would be to choose a type of compression for the video using the `image_transport` ROS package but that depends on the chosen method of communication. Since the robot was currently using only fast laboratory Wi-Fi there was no need for compression.



*Fig. 3.7: Camera image window.*

## 4 Autonomous navigation

It is very difficult to navigate the CLOVER robot into its transportation net because the teleoperator navigates only using a camera with very limited field of view and because there is a lag between a command and its execution which depends on the distance between the robot and the teleoperator. That's why there is a request to make the navigation autonomous. There are many different possibilities for solving this. I have narrowed the options to basically three solutions. The first one is using the visual recognition. My idea was that there would be lights on each pole of the net and the distance and orientation would be calculated based on the position of the lights. For example, the closer together the lights were the further the net was and so on.



*Fig. 4.1: Lights on the net.*

The advantage of this solution is that there is no need to mount any additional sensor. The problem is that the camera is not shielded from dust or mud. It may also have problem with tall grass or other obstacles. Finally, the camera would also have to be oriented towards the net all the time or periodically turn towards the net to determine the robots position.

We could also try to use a scanner. The advantage is that they are reliable. The problem is that they are heavy, thus reducing the UAV flight time, and rather expensive. Given that the robot is carried by UAV in case of an accident we would lose another expensive piece of technology.

Tab. 4.1: Comparison of navigation methods.

Method	Weight of sensor	Reliability	Price
Visual recognition	✓	✗	✓
Laser scanning	✗	✓	✗
IR signal	✓	✓	✓

My other proposal was to use infrared technology. It would work on a principle of an IR beacon and a sensor consisting of multiple phototransistor separated by a barrier. The simplest application of this is a robot going straight for the beacon. The sensor consists of just two phototransistors divided by a barrier.



Fig. 4.2: Robot with two phototransistors (light grey) divided by barrier (brown).

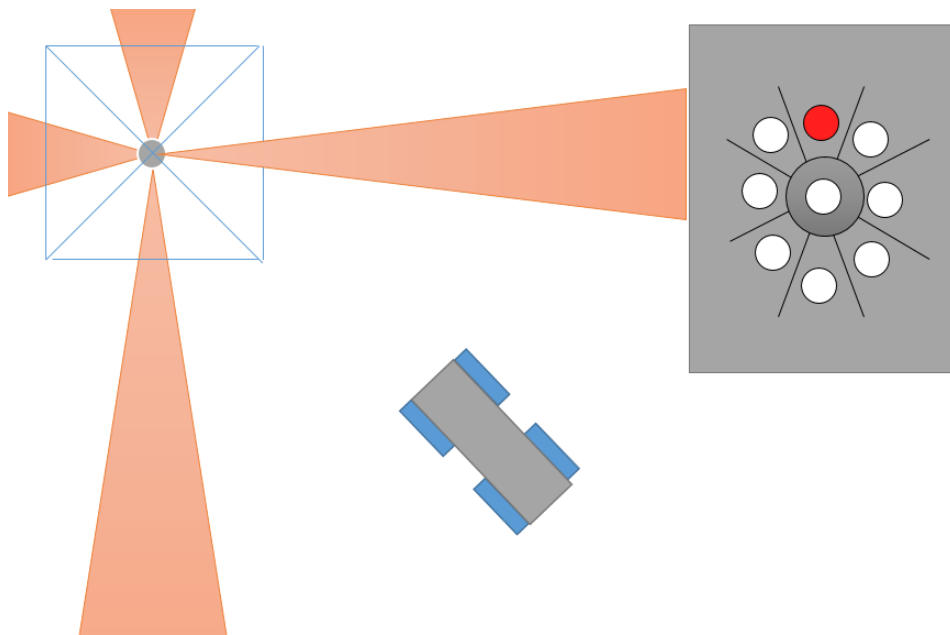
Based on which phototransistor received signal from the beacon the robot would move according to the table 4.1.

Tab. 4.1: Two phototransistor robot behaviour.

Left phototransistor	Right phototransistor	Movement
1	1	Move forward
1	0	Turn left
0	1	Turn right
0	0	Stop (no signal)

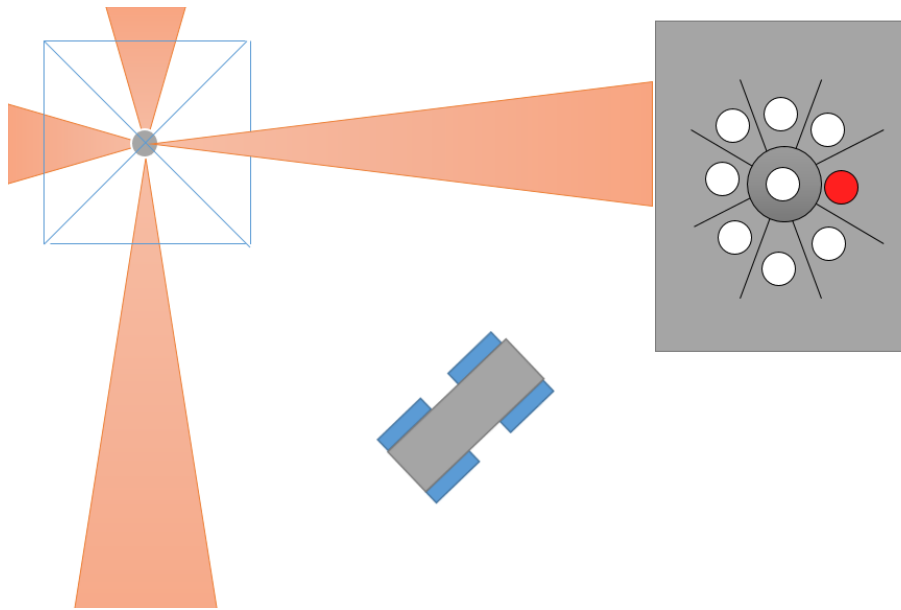
This is only useful when there are no obstacles in the way. Unfortunately, there are poles at the corners of the net that must be avoided. To solve this, I designed a more complex IR sensor and control algorithm.

For this solution, there is a need for two beacons each with different frequency of modulated signal. One will cover 360° around the net and the other one is directional (red area in fig. 4.3). The new sensor consists of multiple phototransistors divided by barrier that accept signal from the sides to determine orientation and one phototransistor surrounded by barrier so that it can only accept signal from above. There will be a need for either two sensors, each for different frequency, or one sensor with two filters.



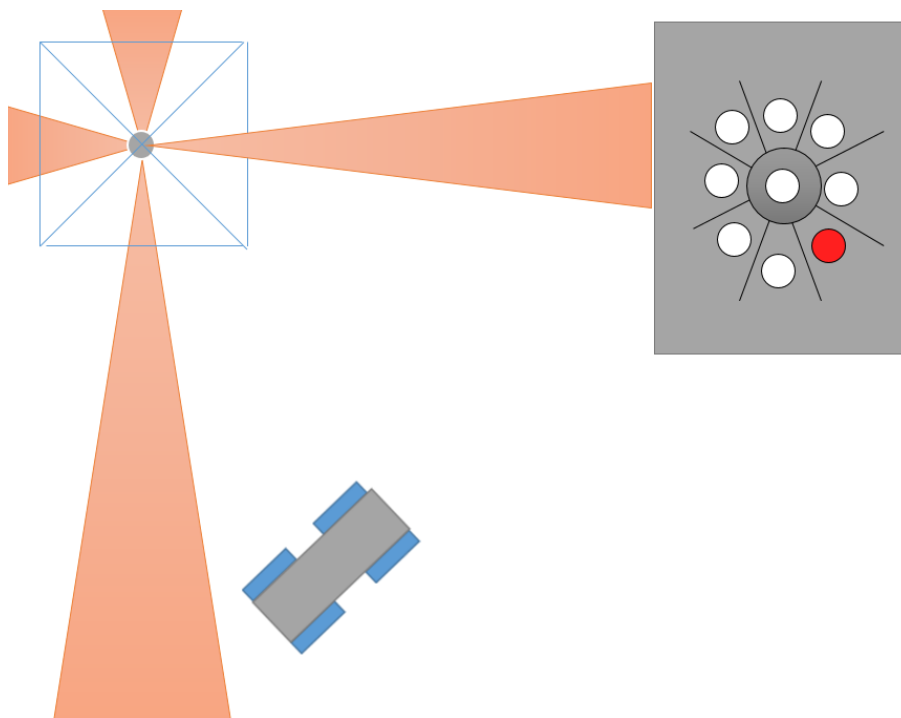
*Fig. 4.3: Stating position.*

The algorithm will first turn the robot either left or right based on the initial position until it is approximately tangent to the imaginary circle with the centre in the beacon and radius of the distance between the robot and the beacon. This corresponds with either the red phototransistor in fig. 4.4 giving logical 1 or the one on the opposite side of the sensor.



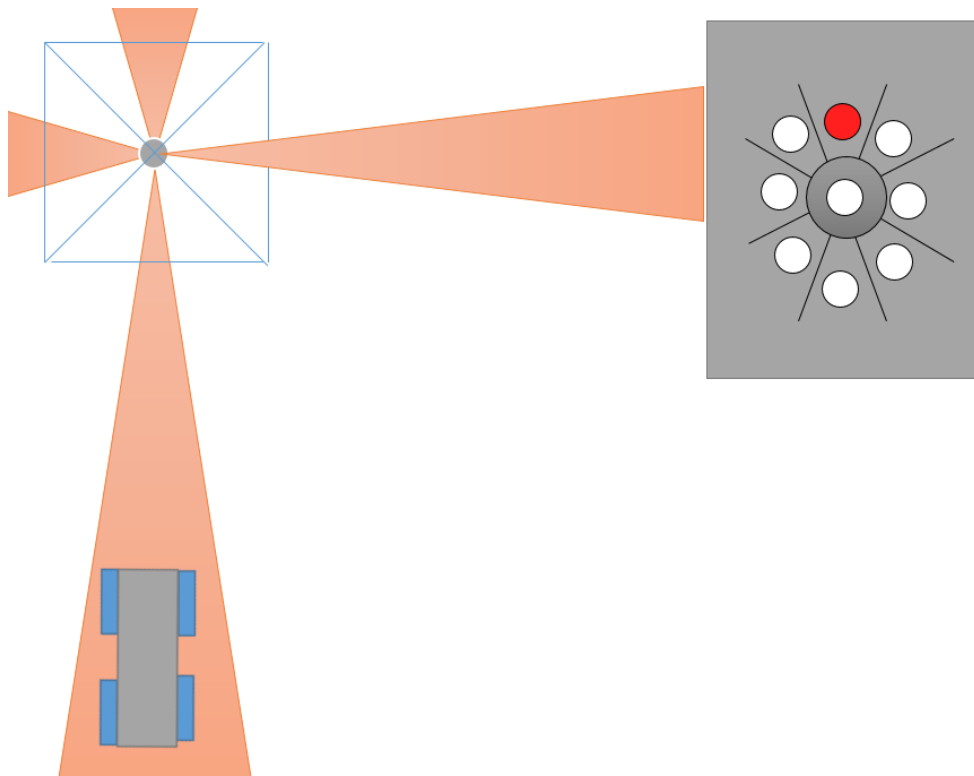
*Fig. 4.4: Initial turn of the robot.*

The robot then starts to move forward for as long as the same phototransistor gives the signal high. When the robot moves too far and the phototransistor changes for example according to the fig. 4.5 the robot then readjusts itself back into orientation shown in fig. 4.4. This can be done multiple times creating a sort of circle around the net until the robot reaches a red area with differently modulated signal.



*Fig. 4.5: Robot before orientation correction.*

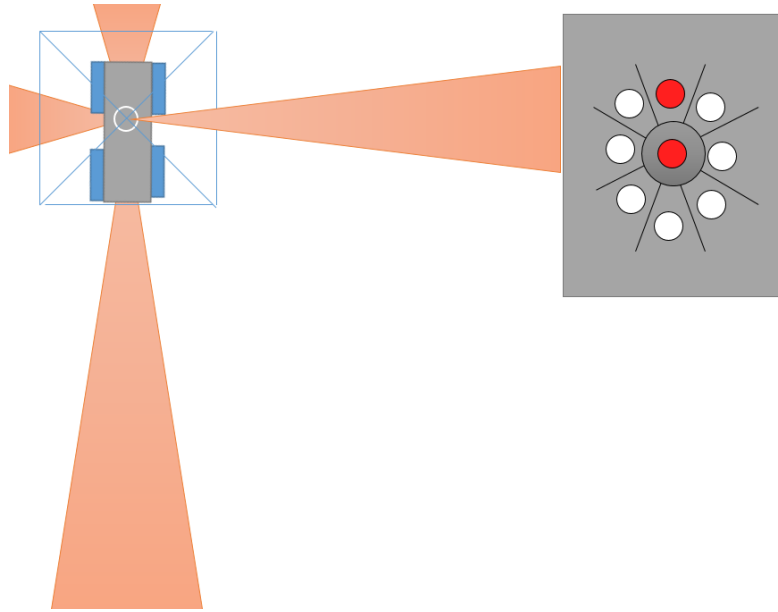
When the robot encounters the area with differently modulated signal the robot turns towards the net and starts moving forward.



*Fig. 4.6: Robot before the final approach.*

Since the sensor can only detect direction and not distance, it was necessary to find another way of knowing that the robot has reached the destination. This is done by the phototransistor in the middle of the sensor which gives logical 1 only when the beacon is above it.

Alternatively, we could add a separate sensor for measuring the distance. But this would add more weight to the robot which is something we should try to avoid. It will be necessary to test whether my proposed solution yields satisfactory results or whether there is a need for additional sensor.



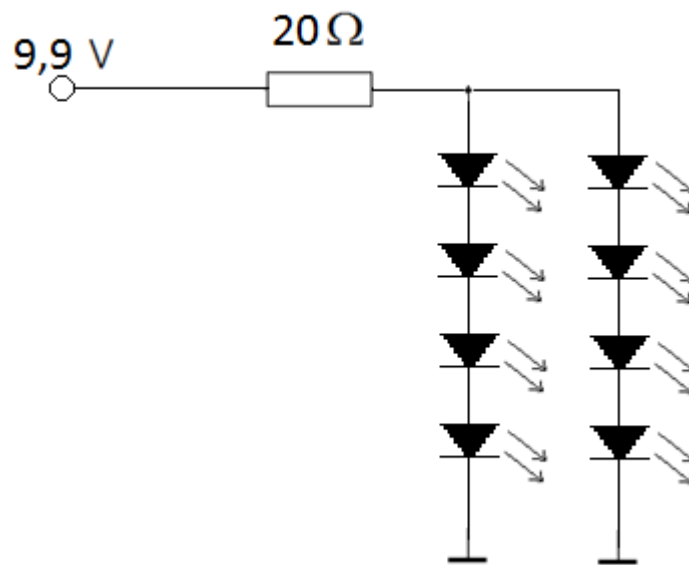
*Fig. 4.7: Final position.*

After the discussion with my supervisor we decided to try the IR approach.



## 5 Implementation of the proposed solution

The first task was to design a beacon that would be hanged on the net and the receiver that would be connected to the Raspberry Pi. I decided to try using unmodulated signal first. The most important parts of the beacon are the IR LEDs. I was searching for LEDs with high intensity and wide angle of radiation. I have chosen TSFF5510 IR LED diodes. For the first test, I used 8 LEDs in serio-parallel configuration.



*Fig. 5.1: IR beacon prototype.*

According to the datasheet the voltage drop of the diode is approximately 1,5 V for 100 mA. To power the diodes, I chose a 9,9 V LiFe battery because it's small while being powerful enough. To determine the value of the resistor I first used the second Kirchhoff's law to gain the resistor voltage

$$U_R = U_B - 4U_D \Rightarrow 9,9 - 4 \cdot 1,5 = 3,9 \text{ V}$$

where:

$U_R$  is the voltage on the resistor

$U_B$  is the battery voltage

$U_D$  is the diode voltage drop

And then the Ohm's law to get the resistance

$$R = \frac{U_R}{2I_D} \Rightarrow \frac{3,9}{2 \cdot 0,1} = 19,5 \Omega$$

where:

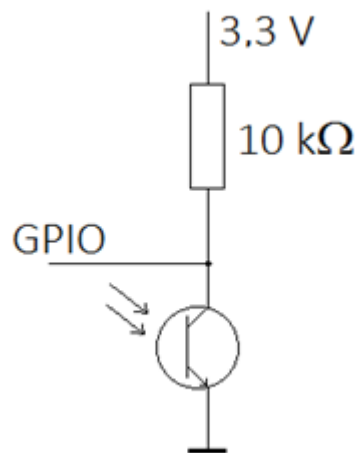
$R$  is the resistance of the required resistor

$U_R$  is the voltage on the resistor

$I_D$  is the current passing through the diode

The closest match was  $20\ \Omega$  resistor. (Vishay, 2012)

For the receiver design, I used OP593B phototransistor with  $10\ \text{k}\Omega$  pull-up resistor.

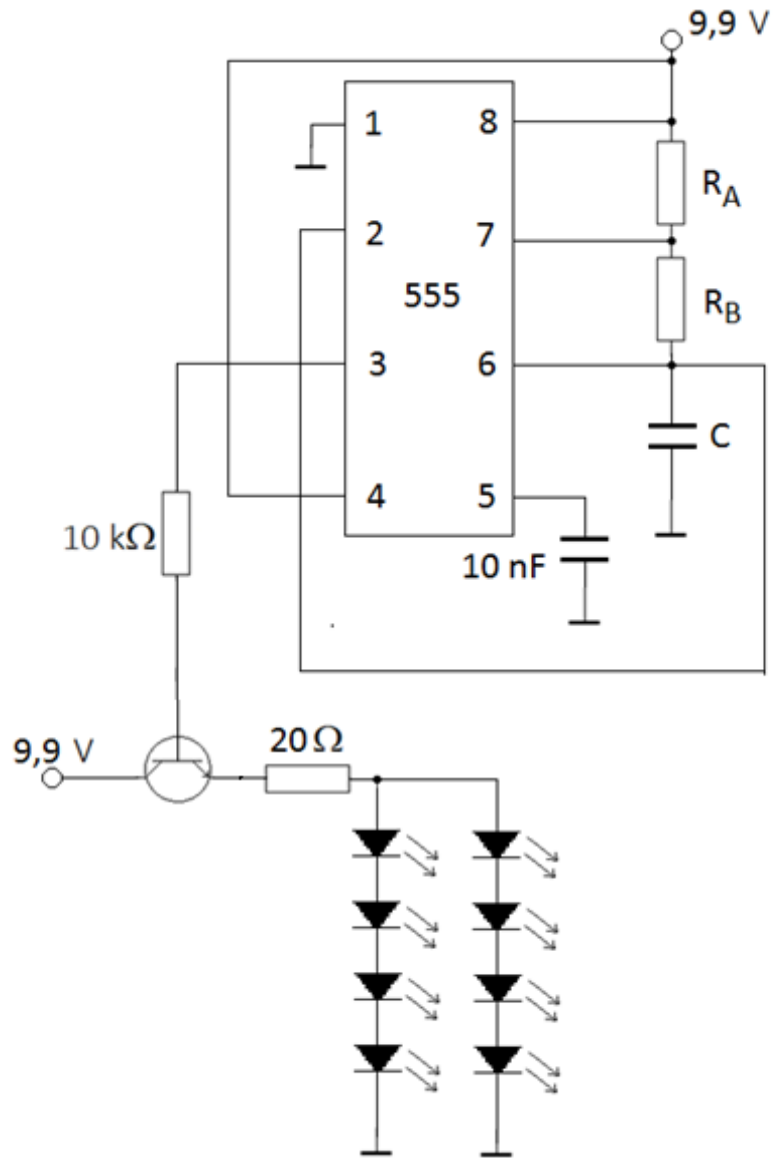


*Fig. 5.2: Phototransistor setup.*

Unfortunately, I was only able to achieve a range of approximately 20 cm with this setup, which is insufficient for robot navigation. To increase the range, I wanted to modulate the signal similarly to the TV remote signal.

## **5.1 Beacon design**

To create a 38 kHz modulated IR signal, I had to modify the beacon. My idea was that the modulation would be created by 555 timer whose output would be connected to the base of the transistor that would switch on and off the LEDs.



*Fig. 5.3: Beacon with modulated signal.*

The oscillation frequency of astable 555 timer is determined by the values of  $R_A$ ,  $R_B$  and  $C$  according to this formula. (Texas Instruments, 2016)

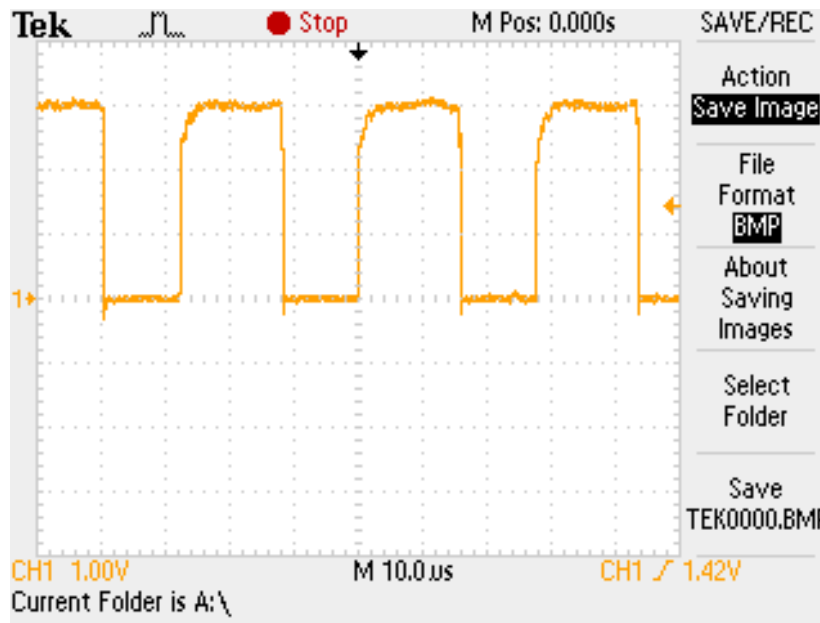
$$f = \frac{1,44}{(R_A + 2R_B)C}$$

where  $f$  is the frequency of the oscillation and  $R_A$ ,  $R_B$  and  $C$  are values of parts shown in fig. 5.3.

I chose the values  $R_A = 600 \Omega$ ,  $R_B = 1,6 \text{ k}\Omega$  and  $C = 10 \text{ nF}$ , which give the following theoretical frequency:

$$f = \frac{1,44}{(R_A + 2R_B)C} \Rightarrow \frac{1,44}{(600 + 2 \cdot 1\,600) \cdot 10^{-8}} = 37\,894,7 \text{ Hz}$$

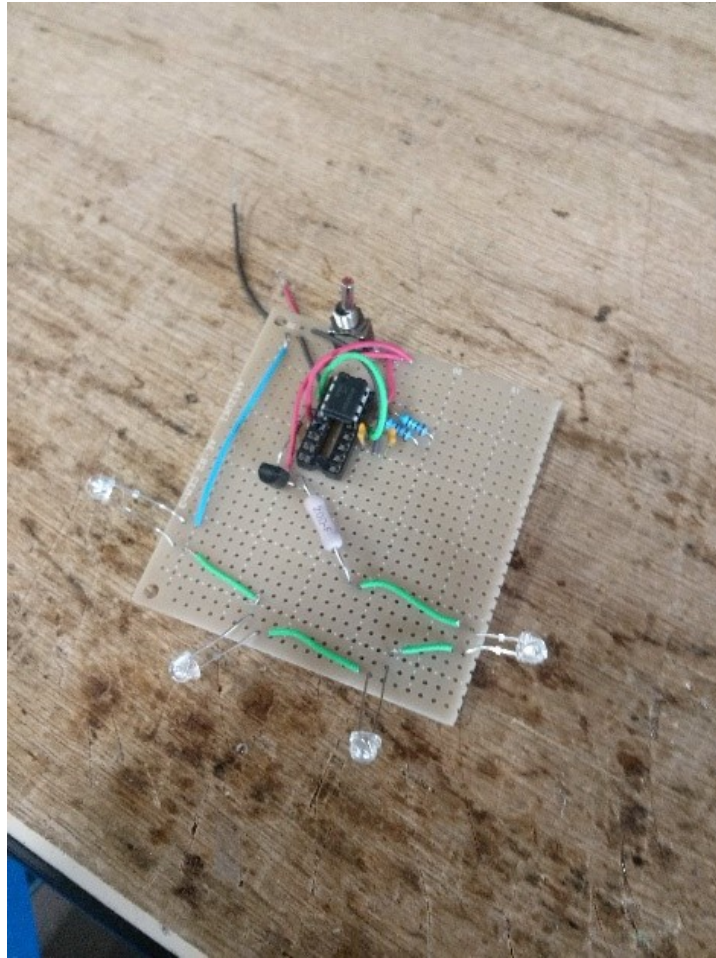
Due to the tolerances of electronic components the real frequency is approximately 36 kHz. This isn't necessarily a problem but by correcting this error a greater range could be achieved. The correction can be achieved for example by putting a potentiometer in serial connection with the resistor and tuning the frequency manually.



*Fig. 5.4: Voltage signal measured on the 20  $\Omega$  resistor.*

As can be seen from fig. 5.5 the circuit generates pulses. The problem is that the collector-emitter voltage drop of the transistor reduces the current flowing through the LEDs to mere 75 mA which, given that the duty is approximately 60 %, is not enough and the range of the IR signal is just over 2 meters.

By reducing the number of LEDs to just 4 in a serial connection I doubled the current to 150 mA which resulted in range of approximately 4 meters. According to the datasheet the LED can handle up 1 A which would increase the range of the signal even further but this would require redesigning the circuit and choosing a more powerful battery. (Vishay, 2012)



*Fig. 5.5: Beacon circuit board.*

## **5.2 Receiver design**

For receiving this modulated signal, I chose IR receiver TSSP4038 that already contains band filter for 38 kHz and demodulator. (Vishay, 2014)

To test this sensor, I set up a circuit shown in fig. 5.6. Without the 33 kΩ resistor the voltage would be 5 V which is over the GPIO 3,3 V limit. With the resistor, the voltage is only 2,5 V which is enough to trigger the logical 1 of the input.

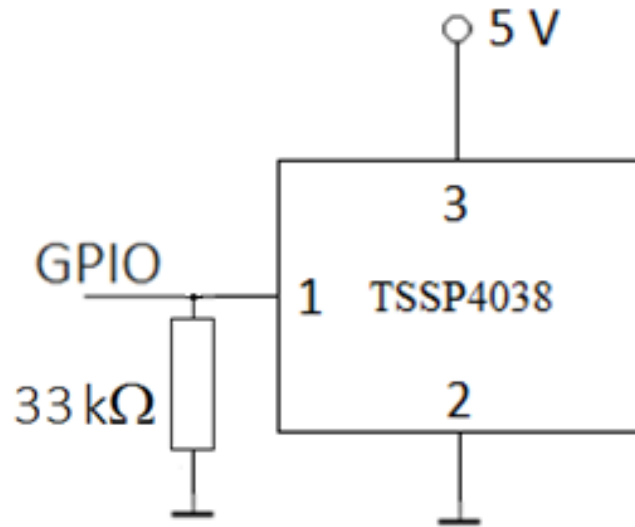


Fig. 5.6: IR sensor test setup.

To test whether the IR receiver works properly I connected the probes of the oscilloscope between the output and ground. Then I pointed a TV remote that operates at the carrier frequency of 38 kHz towards the sensor and pressed random buttons. The result shown at fig. 5.7 confirms that the sensor is connected properly and operates as expected.

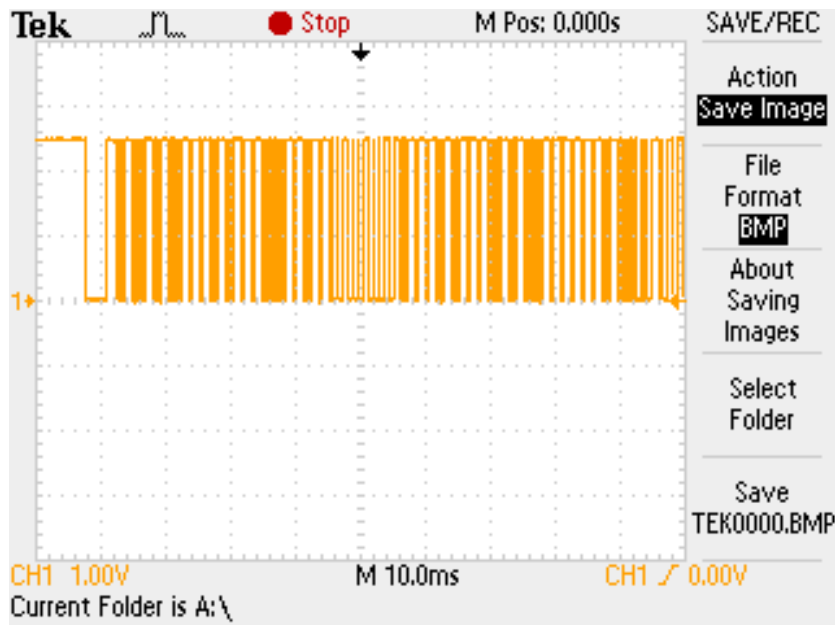
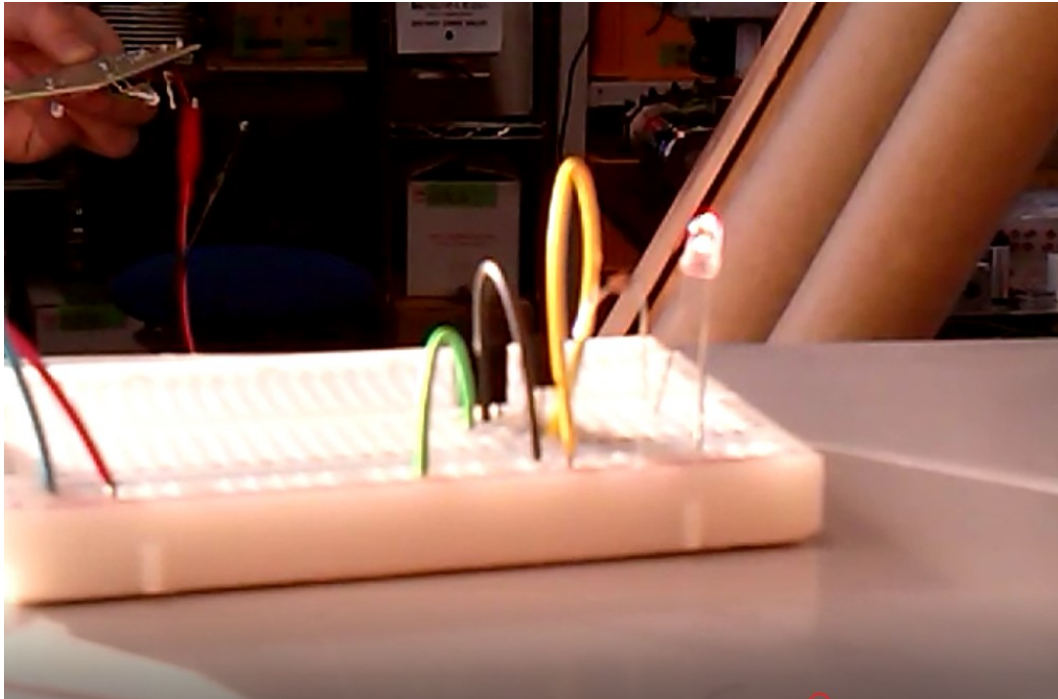


Fig. 5.7: Output of the IR receiver.

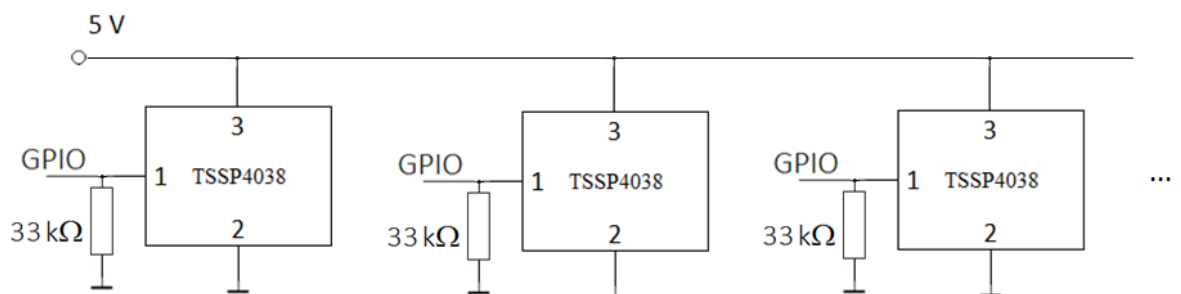
The previous test was conducted inside the laboratory with table lamp as the only source of light. Since the robot operates in the outside it was necessary to test whether

the sun, the biggest IR source we know, would interfere with the sensors operation. I took one IR receiver and connected a regular red LED to the output. Then I put the receiver to the sunlight and pointed the beacon at the receiver. The result is that the LED turns on and off according to the beacon meaning that the receiver works even when exposed to the sunlight.



*Fig. 5.8: Testing the sensor in sunlight.*

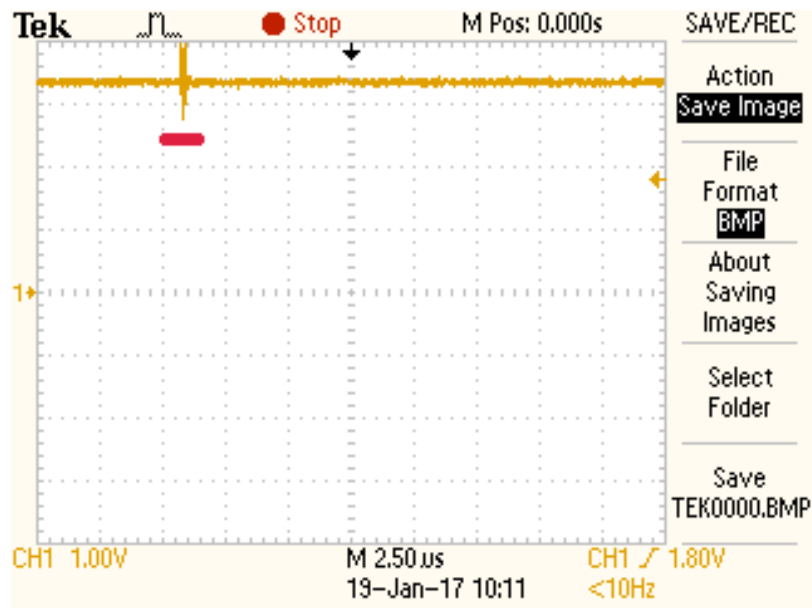
After testing one sensor I connected 9 sensors in parallel. This setup works but there are multiple issues which need addressing.



*Fig. 5.9: Parallel connection of sensors.*

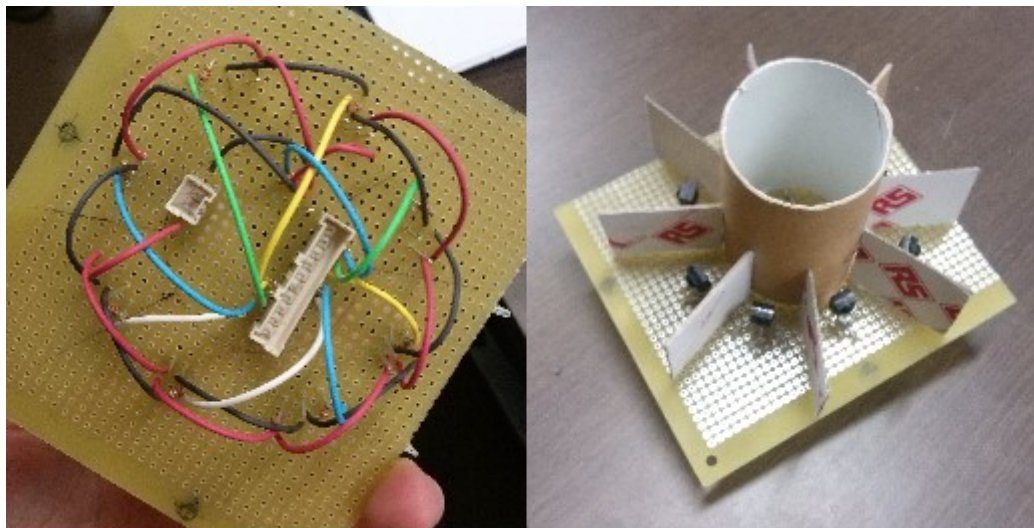
The first one is that for some reason the voltages at the outputs are not equal. The difference however is so small that it doesn't affect functionality. The other problem are pulses that randomly appear at the output. Again, they don't affect functionality but they

may for example reduce service life and this is something that can't be immediately observed.



*Fig. 5.10: Output pulses.*

For creating the barrier, I used a universal circuit board to which I soldered 8 sensors in a circle facing outwards and one in the middle facing upwards. Then I glued pieces of cardboard to the circuit board as barriers. From the bottom, there are two connectors. One is a two-pin connector for 5 V power supply from the voltage regulator and ground. The other one is a nine-pin connector for the GPIO signal wires.



*Fig. 5.11: Assembled sensor bottom (left) and top (right).*



### 5.3 Navigation script

The signal wires from the receiver are connected to GPIO port. The control of the motor is done by publishing speed information on a `cmd_vel` ROS topic. It was therefore necessary to create a program that would read the GPIO port and based on the logical values of the individual pins it would publish translation or rotation speed command on the `cmd_vel` topic.

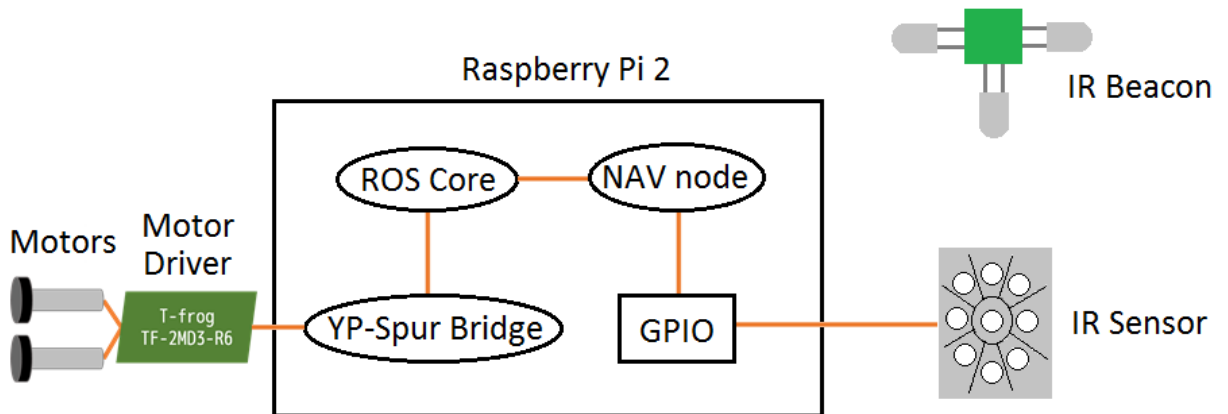


Fig. 5.12: Control system diagram.

I decided to write the program in python because there is a python library called `pigpio` that allows controlling the GPIO pins in Ubuntu. I created a folder called `scripts` inside the `bs_start` directory and I created a file called `nav.py` for the script itself.

At first, I put in a python interpreter directive and import modules for working with ROS in python, GPIO controller, time and Twist messages.

```
#!/usr/bin/env python
import rospy
import time
import pigpio
from geometry_msgs.msg import Twist
```

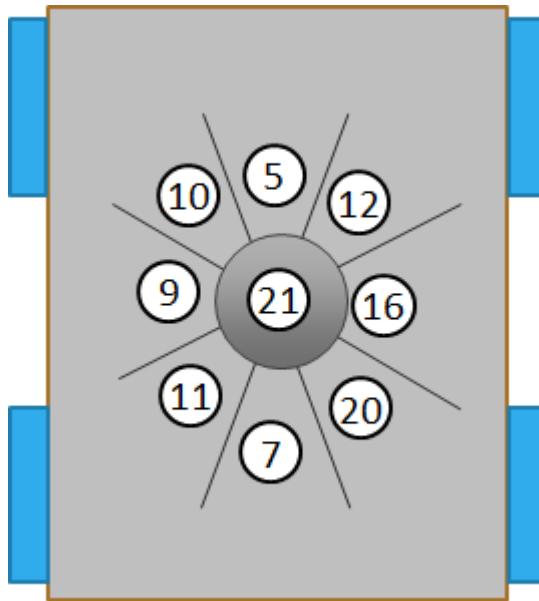
Then I need to set up GPIO pins as inputs. The visual representation of pins can be seen at fig. 5.13

```
gpio = pigpio.pi()
gpio.set_mode(7, pigpio.INPUT)
gpio.set_mode(11, pigpio.INPUT)
```

```

gpio.set_mode(9, pigpio.INPUT)
gpio.set_mode(10, pigpio.INPUT)
gpio.set_mode(5, pigpio.INPUT)
gpio.set_mode(12, pigpio.INPUT)
gpio.set_mode(16, pigpio.INPUT)
gpio.set_mode(20, pigpio.INPUT)
gpio.set_mode(21, pigpio.INPUT)

```



*Fig. 5.13: Sensors with corresponding GPIO pin numbers.*

I also set the internal pull-up resistor to increase the reliability of the system. The pull-ups and not pull-downs because the sensors are inverted to common perception. The sensor is in high state by default and changes to low state when receiving signal.

```

gpio.set_pull_up_down(7, pigpio.PUD_UP)
gpio.set_pull_up_down(11, pigpio.PUD_UP)
gpio.set_pull_up_down(9, pigpio.PUD_UP)
gpio.set_pull_up_down(10, pigpio.PUD_UP)
gpio.set_pull_up_down(5, pigpio.PUD_UP)
gpio.set_pull_up_down(12, pigpio.PUD_UP)
gpio.set_pull_up_down(16, pigpio.PUD_UP)
gpio.set_pull_up_down(20, pigpio.PUD_UP)
gpio.set_pull_up_down(21, pigpio.PUD_UP)

```

Then follows the only function of the program called `nav`. Inside the function there is a definition of the ROS publisher, initialisation of the navigation node and finally a one second delay. The delay is necessary because initialising node takes some time and without it the program does not work properly.

```
pub = rospy.Publisher('/ypspur/cmd_vel', Twist, queue_size=10)
motion = Twist()
print "Start init"
rospy.init_node('navnode')
time.sleep(1)
print "Init done"
```

Then the navigation algorithm itself follows. It is created by a `for` loop because the final approach towards the transportation net is not implemented yet and without it the robot would circle around the beacon indefinitely. After every loop, the script sleeps for 0,1 second. This is programmed to give the motors more time to react to changes and avoid spamming the `cmd_vel` topic. The navigation algorithm is based on `if` conditions. We can see that it first tests whether it should rotate the robot and if not then it starts to move the robot forward. I also tested the opposite method. I made the robot move forward by default and tested if it should stop and rotate. This however resulted in robot slowly getting out of the range of the beacon. The last `if` condition applies when the robot is facing the wrong direction. It turns the robot until other `if` condition applies.

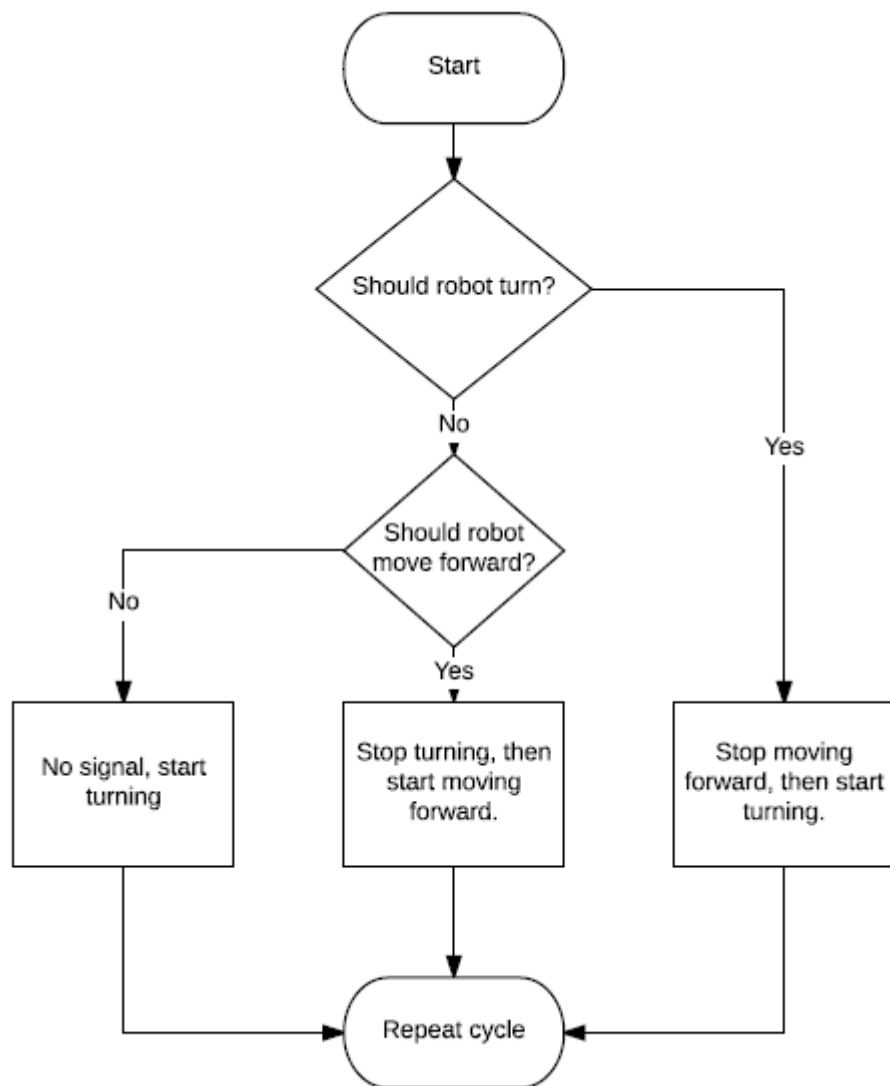
```
for x in xrange(175):
    time.sleep(0.1)
    if gpio.read(20) == 1:
        motion.angular.z = 0
        print "Stop turn"
        if gpio.read(16) == 1:
            motion.linear.x = 0
            print "Stop forward"
        if gpio.read(16) == 0:
            motion.linear.x = -0.2
            print "Forward"
    if gpio.read(20) == 0:
```

```

motion.angular.z = -1
print "Turn"
if (gpio.read(16) == 1 and gpio.read(20) == 1):
    motion.angular.z = -1
    print "Wrong orientation -> turning"
pub.publish(motion)

```

The visual representation of the loop can be seen at fig. 5.14.



*Fig. 5.14: Flow chart of the navigation loop.*

After the for loop end, there are commands to stop the robot and notify the user about it.

```

motion.angular.z = 0

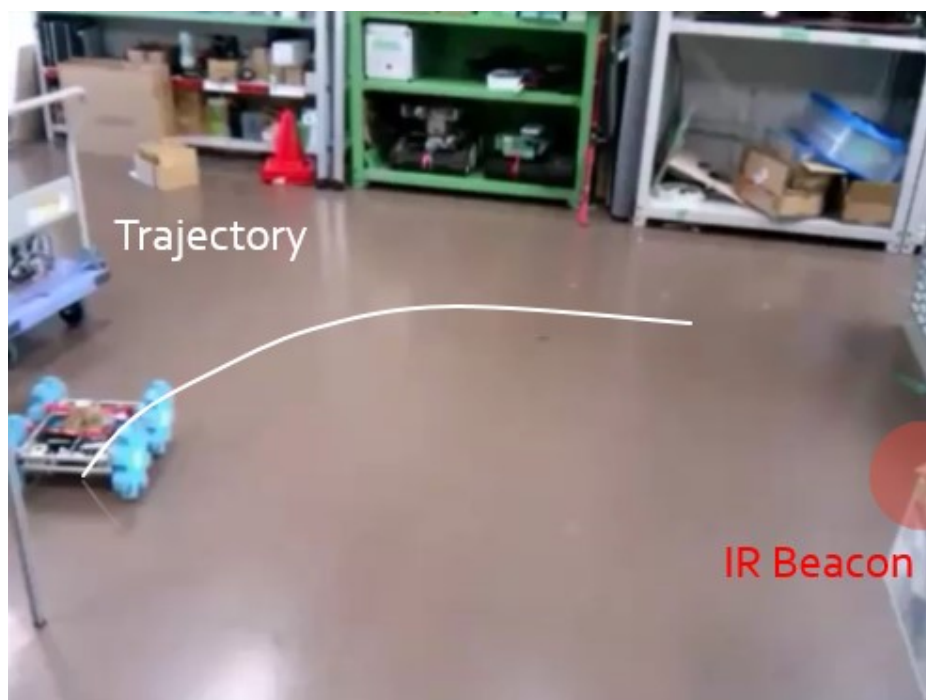
```

```
motion.linear.x = 0
pub.publish(motion)
print "end"
```

The final part of the script consists of calling the navigation function or quitting the program

```
if __name__ == '__main__':
    try:
        nav()
    except rospy.ROSInterruptException:
        motion.angular.z = 0
        motion.linear.x = 0
        pub.publish(motion)
```

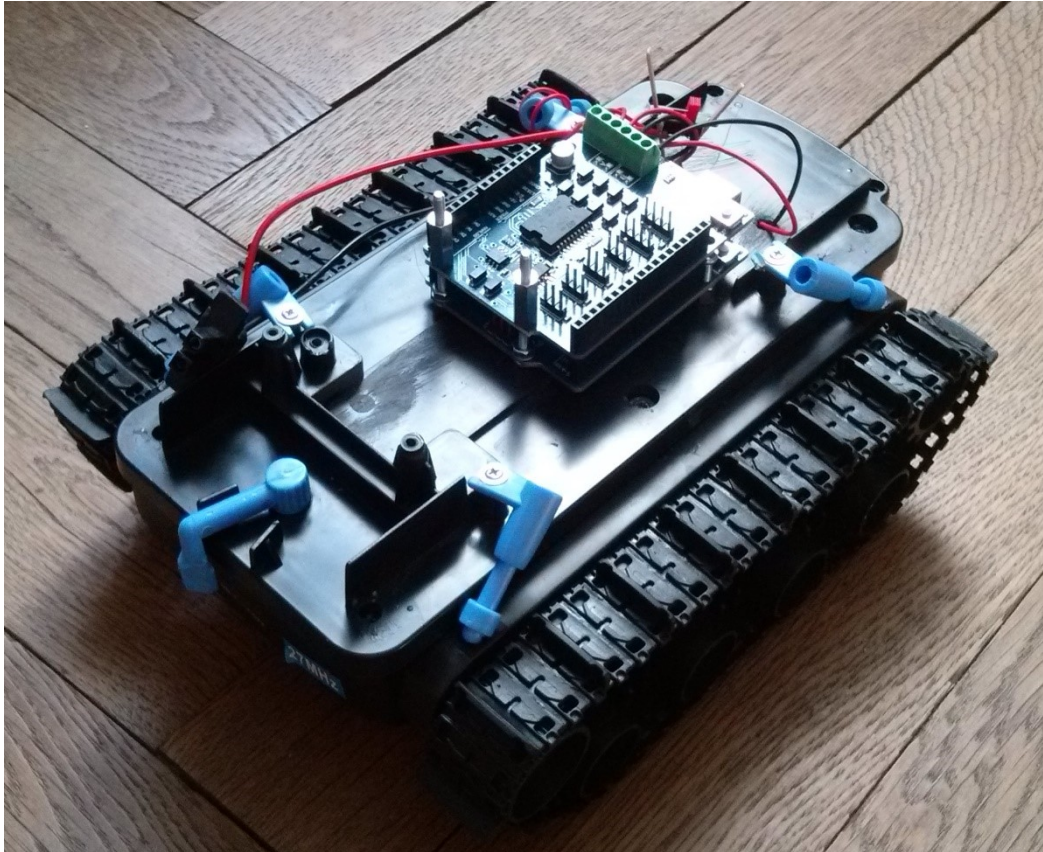
After launching the script, the robot turns until it's tangent to the imaginary circle with the center in the beacon and then start to circle the beacon as shown in fig. 6.4.



*Fig. 5.15: Robot trajectory based on the algorithm above.*

## 6 IR navigation of the Arduino robot

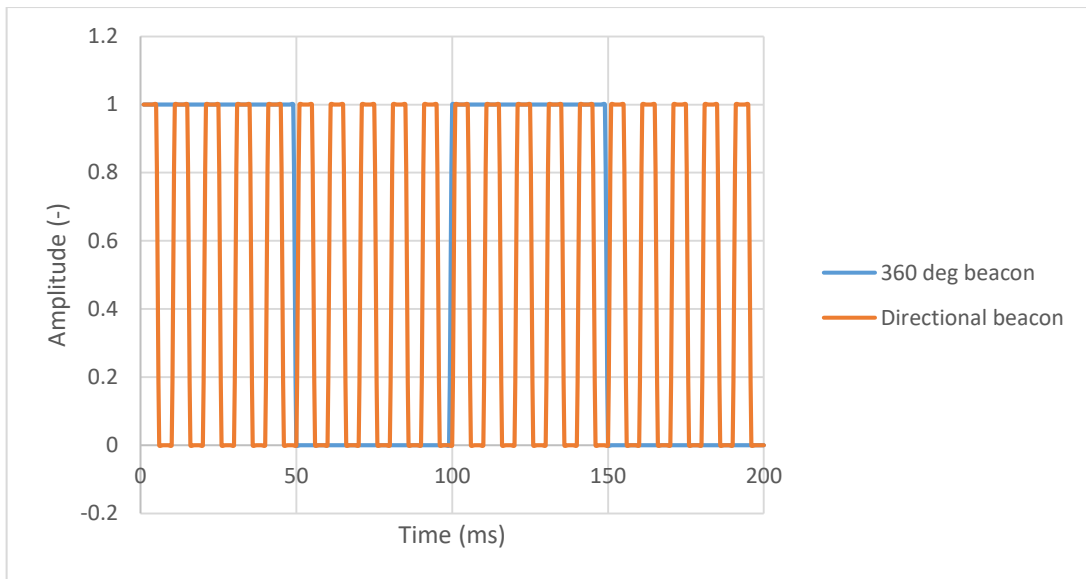
In this chapter, I describe the continuation of the IR navigation project, that I conducted after returning to the Czech Republic from Japan. Since I left the CLOVER robot in Japan, I had to create a new vehicle to finish the autonomous navigation. I decided to use a tracked RC car from which I removed the electronics and left only motors and chassis. To drive the motors, I used Arduino Uno with motor shield.



*Fig. 6.1: Arduino robot.*

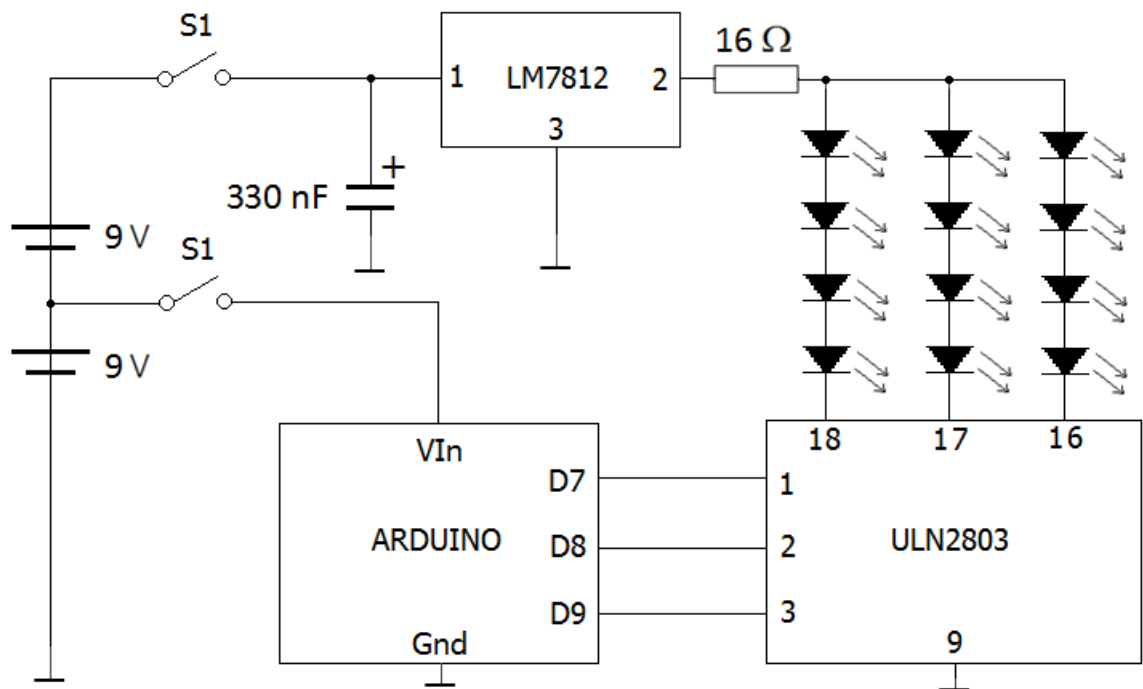
### 6.1 Infrared beacon

Except the CLOVER robot I also left behind the IR beacon and receiver. This time I developed a complete beacon as described in chapter 4 (including red areas). To achieve having two distinctive areas I had to create two different kind of signals. My solution was to emit pulses of 38 kHz signal with different frequencies. The final beacon uses 10 Hz pulses in areas where the robot circulates the beacon and 100 Hz pulses in areas where the robot turns and makes the final approach.



*Fig 6.2: Pulses emitted by the beacon.*

Instead of using a 555 timer to generate 38 kHz pulses and then another two timers to generate 100 Hz and 10 Hz pulses, I decided to use Arduino Nano which comes equipped with three timers. This beacon has 12 LEDs divided into 3 branches. Each branch can be controlled from Arduino through ULN2803 transistor array. The voltage is supplied from two 9V batteries and regulated by L7812 voltage regulator.



*Fig. 6.3: Beacon schematics.*

The beacon is soldered on universal PCBs which are attached to each other by M3 screws. The first PCB has ports for Arduino so it can be removed if necessary. The second contains LEDs that are arranged so that they cover the area of 360 degrees. The last one has the remaining electronic parts.

The program running in Arduino uses three variables. One for generating 38 kHz modulated signal, other two for creating pulses of 10 Hz and 100 Hz.

```
boolean toggle10Hz = 0;
boolean toggle38kHz = 0;
boolean toggle100Hz = 0;
```

Then next is the setup function which run only once after the Arduino boots and where digital pins 7, 8, 9 are set as outputs

```
void setup() {
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
```

and then bits of the timers are set. The following code is for timer 0 which has a frequency of 200 Hz. Timer 1 has a frequency of 20 Hz and differs from timer 0 in the value of compare match register OCR1A = 780. Timer 2 creates the modulated signal of 76 kHz and has compare match register OCR2A = 210, also it has TCCR2B |= (1 << CS20) for no prescaler.

```
//set timer0 interrupt at 200 Hz
TCCR0A = 0;// set entire TCCR2A register to 0
TCCR0B = 0;// same for TCCR2B
TCNT0 = 0;//initialize counter value to 0
// set compare match register for 200 Hz increments
OCR0A = 77;// = (16*10^6) / (200*1024) - 1 (must be <256)
// turn on CTC mode
TCCR0A |= (1 << WGM01);
// Set CS01 and CS00 bits for 1024 prescaler
TCCR0B |= (1 << CS02) | (1 << CS00);
// enable timer compare interrupt
TIMSK0 |= (1 << OCIE0A);
```



The frequency of the timers must be a double of the frequency of the required signal because we need a square signal with a 50% duty cycle. Every timer has similar interrupt service routine, only the variable differs.

```
ISR(TIMER0_COMPA_vect) {
    if (toggle100Hz){
        toggle100Hz = 0;
    }
    else{
        toggle100Hz = 1;
    }
}
```

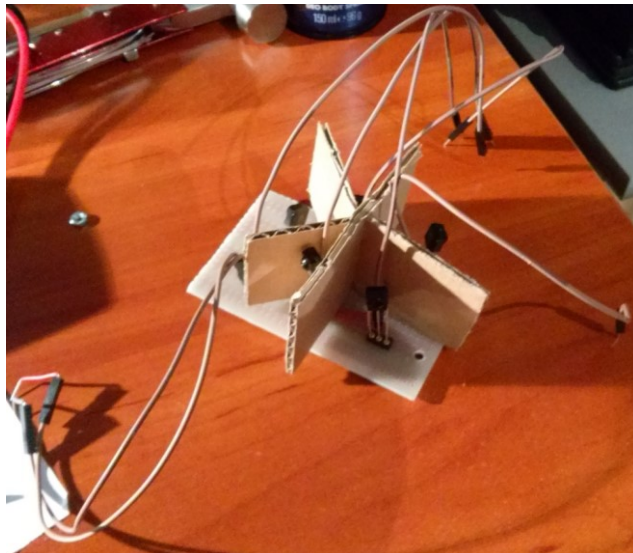
Finally, there is the main loop function, where pins are set according to the timers. The pin 7 is responsible for 100 Hz signal while pins 8 and 9 are responsible for 10 Hz signal.

```
void loop() {
    if (toggle100Hz){
        if (toggle38kHz){
            digitalWrite(7, HIGH);
        }
        else{
            digitalWrite(7, LOW);
        }
    }
    else{
        digitalWrite(7, LOW);
    }
    if (toggle10Hz){
        if (toggle38kHz){
            digitalWrite(8, HIGH);
            digitalWrite(9, HIGH);
        }
        else{
            digitalWrite(8, LOW);
        }
    }
}
```

```
        digitalWrite(9, LOW);
    }
}
else{
    digitalWrite(8, LOW);
    digitalWrite(9, LOW);
}
}
```

## 6.2 Infrared receiver

Not much has changed in the design of the receiver as it still consists of TSSP4038 IR sensors connected in parallel. The main difference is that I reduced the amount of TSSP4038 sensors from 9 to 5. The rear sensor served no purpose and thus was removed first. The front left and front right sensors were removed because I thought that left and right sensors could cover their angle as well. After testing I conclude that the navigation works without them but I would recommend putting them back for increased precision. Finally, the sensor pointing upwards was removed, because I found that when the robot is in its final location, all the sensors activate. I not sure if this is because they have such a wide angle of transmission or the IR signal reflects from the PCB into the sensor.



*Fig. 6.4: IR receiver.*

### 6.3 Navigation algorithm

The navigation script starts with a declaration of variables for the motor control pins. The motor shield also supports current sensing on analog pins but these are not used.

```
const int dirA = 12;    // direction of motor A
const int dirB = 13;    // direction of motor B
const int pwmA = 3;     // duty cycle of motor A pwm
const int pwmB = 11;    // duty cycle of motor B pwm
const int brkA = 9;     // brake of motor A
const int brkB = 8;     // brake of motor B
```

Then the pins for IR receiver are set. Currently there is enough pins for the IR receiver but if the number of sensors increases, it will be necessary to use analog pins as digital.

```
const int forw = 6;     // forward pin
const int lfor = 7;     // left forward pin
const int rfor = 5;     // right forward pin
const int lbac = 10;    // left back pin
const int rbac = 4;     // right back pin
```

Finally, there are variables that are used for navigation but do not have a pin associated with them. First five variables are used to record the state of the digital pins declared above at the beginning of the cycle. Next three are used to record the state of the digital pins at the end of the cycle so that they can be compared with new values in the next cycle. Variable `indexPulse` is set by the timer with frequency of 10 Hz. Variable `pulse` is used for counting pulses from the beacon and is reset every time the `indexPulse` variable is 1. The last variable stores the stage of the autonomous navigation (rotating, circulating beacon, final approach, stopping).

```
int fo, lf, rf, lb, rb;
int lfold = 1;
int foold = 1;
int rfold = 1;
int indexPulse = 0;
int pulse = 0;
int stage = 0;
```

The setup function first sets the motor pins as outputs and receiver pins as inputs, then unbrakes the robot.

```
void setup() {  
  // set up motor pins as outputs  
  pinMode(dirA, OUTPUT);  
  pinMode(dirB, OUTPUT);  
  pinMode(pwmA, OUTPUT);  
  pinMode(pwmB, OUTPUT);  
  pinMode(brkA, OUTPUT);  
  pinMode(brkB, OUTPUT);  
  
  // set up receiver pins as inputs  
  pinMode(forw, INPUT);  
  pinMode(lfor, INPUT);  
  pinMode(rfor, INPUT);  
  pinMode(lbac, INPUT);  
  pinMode(rbac, INPUT);  
  
  // unbrake robot  
  digitalWrite(brkA, LOW);  
  digitalWrite(brkB, LOW);
```

The last part is setting up the timer that generates index pulse at 10 Hz

```
TCCR1A = 0; // set entire TCCR1A register to 0  
TCCR1B = 0; // same for TCCR1B  
TCNT1 = 0; // initialize counter value to 0  
// set compare match register for 1hz increments  
OCR1A = 1560; // = (16*10^6) / (10*1024) - 1 (must be <65536)  
// turn on CTC mode  
TCCR1B |= (1 << WGM12);  
// Set CS12 and CS10 bits for 1024 prescaler  
TCCR1B |= (1 << CS12) | (1 << CS10);  
// enable timer compare interrupt  
TIMSK1 |= (1 << OCIE1A);
```

The interrupt service routine is simply setting the indexPulse variable to 1. It is reset to 0 at the end of if function in the main loop.

```
ISR(TIMER1_COMPA_vect) {  
    indexPulse = 1;  
}
```

Next there are function whose purpose is moving the robot. They are a combination of setting the direction pins and setting the pwm pins.

```
void nomove() {  
    analogWrite(pwmA, 0);  
    analogWrite(pwmB, 0);  
}
```

```
void forward() {  
    digitalWrite(dirA, HIGH);  
    digitalWrite(dirB, LOW);  
    analogWrite(pwmA, 235);  
    analogWrite(pwmB, 255);  
}
```

```
void backward() {  
    digitalWrite(dirA, LOW);  
    digitalWrite(dirB, HIGH);  
    analogWrite(pwmA, 235);  
    analogWrite(pwmB, 255);  
}
```

```
void left() {  
    digitalWrite(dirA, LOW);  
    digitalWrite(dirB, LOW);  
    analogWrite(pwmA, 235);  
    analogWrite(pwmB, 255);  
}
```

```

void right() {
    digitalWrite(dirA, HIGH);
    digitalWrite(dirB, HIGH);
    analogWrite(pwmA, 235);
    analogWrite(pwmB, 255);
}

```

Finally, there is the loop function that contains switch statement which wraps the entire navigation algorithm. The delay of 3 ms was experimentally chosen as optimal.

```

void loop() {
    switch (stage) {
        // stages of autonomous navigation
    }
    delay(3);
}

```

The first case of the switch statement corresponds with the stage of the navigation, where the robot turns so it is tangent to an imaginary circle with the center in the beacon. This requires all the sensors to check for signal. Based on which sensor returns positive signal, the robot turns either clockwise or counter-clockwise and the algorithm advances to the next stage. In case that the robot receives no signal (it is too far from the beacon) it just does not move. It might be beneficial to implement some form of signalling when no signal is received.

```

case 0: {
    //rotation around robot's axis
    fo = digitalRead(forw);
    lf = digitalRead(lfor);
    rf = digitalRead(rfor);
    lb = digitalRead(lbac);
    rb = digitalRead(rbac);

    if (rf==0) {
        nomove();
        stage = 1;
    }
}

```

```

else if (lf==0){
    nomove();
    stage = 2;
}
else if ((fo==0) || (lb==0)){
    left();
}
else if (rb==0){
    right();
}
break;
}

```

The next stage deals with circulating the beacon. In stage 1, the robot circulates clockwise and then skips to stage 3, because stage 2 serves for counter-clockwise navigation. During this stage, only two sensors are active. This helps reducing confusion caused by reflection into other sensors. The last if function checks the number of pulses each 100 ms to determine whether the robot should turn to take the final approach or not.

```

case 1:{
    //clockwise rotation around beacon
    rf = digitalRead(rfor);
    rb = digitalRead(rbac);
    if (rb==0){
        right();
    }
    else if (rf==0){
        forward();
    }
    if (rf!=rfold){
        pulse++;
    }
    if (indexPulse){
        if (pulse>8){
            right();

```

```

        stage = 3;
    }
    pulse=0;
    indexPulse=0;
}
rfold = rf;
break;

```

The case 2 statement is same as case 1 except left pins and variables are used. The stage 3 is where the robot makes the final approach towards the transportation net. When the robot turns towards the net, it stops with a little deviation. The robot starts moving forward, but the front sensor quickly loses signal. The algorithm then checks if front left or right sensor has signal and turns respectively until the front sensor receives signal again. This means that the final approach is not a straight line but rather a general curve.

```

case 3:{
    //translation towards beacon
    nomove();
    fo = digitalRead(forw);
    lf = digitalRead(lfor);
    rf = digitalRead(rfor);
    lb = digitalRead(lbac);
    rb = digitalRead(rbac);
    if (fo!=foold){
        pulse++;
    }
    if (indexPulse){
        if (pulse>7){
            forward();
        }
        else if(lf == 0){
            left();
        }
        else if(rf == 0){
            right();
        }
    }
}

```



```
    }
    pulse=0;
    indexPulse=0;
}
foold = fo;
//stopping under the beacon
if ((fo==0) && (lf==0) && (rf==0) && (lb==0) && (lb==0)){
    stage = 4;
}
break;
```

The final stage contains delay, because the robot would otherwise stop a bit early.  
Then the robot stops.

```
case 4:{
    delay(1500);
    nomove();
}
```

## Conclusion

The purpose of this research was to improve the volcano observation robot by creating a method of autonomous navigation into the robot's transportation net. At first an old robot that was not needed anymore was disassembled to provide the Maxon brushed motors, aluminium skeleton and belts to recreate the original CLOVER robot. Other removed parts were identified but were either useless for current function of the robot or obsolete.

A control system based on Raspberry Pi 2 was implemented. The microcomputer uses Ubuntu 14.04 as an operating system and has ROS Indigo installed. The Maxon motors are controlled by the T-FROG motor driver that is connected to the Raspberry Pi over the USB. Communication is then based on publishing on the ROS topics.

To allow the navigation of the robot without the direct line of sight an iBuffalo webcam was added to the robot. It connects to the Raspberry Pi's USB port. For getting the camera image a `usb_cam` ROS driver was used, that publishes camera feed over the `image_raw` topic into the `rviz` visualisation software.

Next, various methods for autonomous navigation were discussed and it was decided to try using infrared signal to navigate the robot. An algorithm for the navigation was developed. The designed navigation consists of three steps. First the robot turns until it's tangent to an imaginary circle with the centre in the beacon. Then the robot starts following the circle until it arrives to a position where it can avoid the net structure. Finally, the robot turns towards the net and moves forward until it is inside.

For the beacon-sensor pair, two versions were created. The first version used unmodulated signal and consisted of a beacon made of 8 infrared LEDs connected to a 9,9 V LiFe battery and a receiver made of phototransistors with 10 k $\Omega$  pull-up resistors. The range of this solution was 20 cm which is not enough to navigate the robot.

The second version uses 38 kHz modulated signal generated by a 555 timer circuit. The beacon was made by adapting the previous variant and the problem is that it currently operates at approximately 15 % of possible power because the LEDs can be powered by up to 1 A at this frequency. The receiver part is made of 9 TSSP4038 IR sensors that are connected parallelly and whose output connects to GPIO port of the Raspberry Pi. The range of the signal is 4 meters.

Next a python script was created that reads the GPIO values using the pigpio library and based on the algorithm previously designed publishes on the cmd\_vel ROS topic. So far, the script enables the robot to circle around the beacon which was a crucial step in the navigation algorithm due to its difficulty.

After returning to the Czech Republic a new vehicle based on Arduino Uno was created. There was also a need for a new beacon, that this time supports entire navigation procedure and uses Arduino Nano as a core. The receiver still uses TSSP4038 IR sensor, but now only five of them. The Arduino is able to successfully navigate into its transportation net.

## Literature

Askubuntu. How to access a usb flash drive from the terminal? [online]. 2012 [accessed 2. 12. 2016]. Available at: <http://askubuntu.com/questions/37767/how-to-access-a-usb-flash-drive-from-the-terminal-how-can-i-mount-a-flash-driv>

Axis. Etrax 100LX. [online]. 2016 [accessed 18. 11. 2016]. Available at: [http://www.axis.com/files/datasheet/ds\\_etra\\_x\\_mcm\\_4p16\\_21724\\_en\\_0505\\_lo.pdf](http://www.axis.com/files/datasheet/ds_etra_x_mcm_4p16_21724_en_0505_lo.pdf)

FieldRoboticsLaboratory. [Zion+CLOVER] : A basic experiment for deploying and retracting a UGV by a UAV. [online]. 10. 11. 2016 [accessed 16. 11. 2016]. Available at: <https://www.youtube.com/watch?v=-dhlu7JnZSc>

GitLab. BeginnersSeminar関連のセットアップ (Beginner's seminar related setup). [online]. 6. 2016 [accessed 11. 11. 2016]. Available at: [http://172.20.0.20:8000/ALL/BeginnersSeminar/wikis/tutorial/bs\\_setup](http://172.20.0.20:8000/ALL/BeginnersSeminar/wikis/tutorial/bs_setup)

GitLab. 遠隔操作してみる (Try remote control). [online]. 9. 2016 [accessed 11. 11. 2016]. Available at: <http://172.20.0.20:8000/ALL/BeginnersSeminar/wikis/work/RC>

Maxon. Encoder MR. [online]. 2016a [accessed 14. 10. 2016]. Available at: [http://www.maxonmotorusa.com/medias/sys\\_master/root/8821073346590/16-390-391-EN.pdf](http://www.maxonmotorusa.com/medias/sys_master/root/8821073346590/16-390-391-EN.pdf)

Maxon. Planetary gearhead GP 22 C. [online]. 2016b [accessed 14. 10. 2016]. Available at: [http://www.maxonmotorusa.com/medias/sys\\_master/root/8821069119518/16-331-332-EN.pdf](http://www.maxonmotorusa.com/medias/sys_master/root/8821069119518/16-331-332-EN.pdf)

Maxon. RE – max 24. [online]. 2016c [accessed 14. 10. 2016]. Available at: [http://www.maxonmotorusa.com/medias/sys\\_master/root/8813532250142/14-162-EN.pdf](http://www.maxonmotorusa.com/medias/sys_master/root/8813532250142/14-162-EN.pdf)

Panasonic. CY-RC90KD. [online]. 2016 [accessed 18. 11. 2016]. Available at: <http://panasonic.jp/car/navi/products/camera/CY-RC90KD.html>

ROS Wiki. Configuring and Using a Linux-Supported Joystick with ROS. [online]. 12. 5. 2016 [accessed 11. 11. 2016]. Available at: <http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick>

ROS Wiki. Installation. [online]. 23. 5. 2016 [accessed 11. 11. 2016]. Available at: <http://wiki.ros.org/ROS/Installation>

Texas instruments. LM555 Timer (Rev. D). [online]. 2016 [accessed 12. 1. 2017]. Available at: <http://www.ti.com/lit/ds/symlink/lm555.pdf>

VirtualBox. No image with USB 2.0 webcams. [online]. 2014 [accessed 28. 11. 2016]. Available at: <https://www.virtualbox.org/ticket/242>

Vishay. TSFF5510 datasheet. [online]. 2012 [accessed 4. 11. 2017]. Available at: <http://docs-asia.electrocomponents.com/webdocs/0dea/0900766b80dea07e.pdf>

Vishay. TSSP4038 datasheet. [online]. 2014 [accessed 5. 11. 2017]. Available at: <http://docs-asia.electrocomponents.com/webdocs/1180/0900766b811803e1.pdf>

## Appendix

A Diploma from STOČ Ostrava competition

Content of the CD:

- File *thesis.pdf*