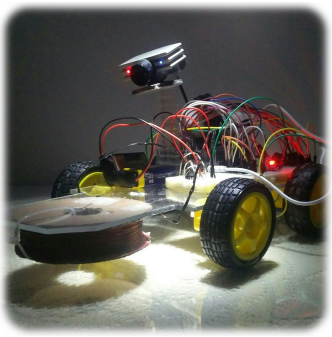

# Autonomous Metal Detector Robot

**Matteo Berti**; 889889

Laboratory of Making
Master's Degree in Computer Science, Department of Computer Science and Engineering, UNIBO

*August 27, 2020*

**Abstract:** The goal of the project was to design, program and build a metal detector robot which can autonomously explore an area and return a feedback to the user. The project consists of two main parts: the *metal detector* module and the *exploratory robot*. These parts were designed, programmed, and tested separately and then joined together.

**Keywords:** robot, metal detector, raspberry pi, arduino, making.

## 1. Metal Detector

There are many metal detector circuits' designs, most of them require advanced knowledge in electronic systems. For this project I used a simple but effective design for the metal detector module.

The main idea is to create a square wave which sends pulses to an air core coil. These pulses produce an alternating magnetic field around the coil, which is proportional to the current it carries. Having an air core the coil inductance is unaffected by the current moving through. When a metal is brought near the coil, eddy current creates a magnetic field in the metal object that opposes the change in the magnetic field that created it, and thus react back on the source of the magnetic field, slowing down the pulses frequency.

### 1.1. Circuit design

There are many ways to create a square wave, I decided to use the **NE555P** which is a very common and cheap integrated circuit with many applications. In **Figure 1** is shown the internal circuit of the IC 555, it is very simple but quite powerful.

In **Figure 2** is shown the circuit designed for the metal detector. In order to create a square wave I chose the **astable** mode which is a common configuration for the IC 555. Connecting the output pin with the threshold pin a 50% duty cycle square wave can be obtained, and depending on the LC circuit impedance the pulses frequency varies.

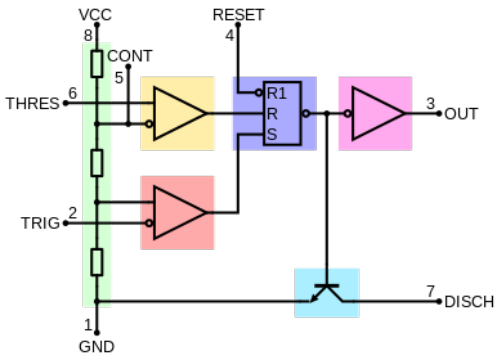


**Figure 1:** IC 555 internal circuit

### 1.2. Coil design

The coil design requires care, being an important feature of the metal detector module. The size of the coil can influence the detection depth and sensitivity. The larger the coil, the more ground it can cover but with less sensitivity. I decided to make my own coil using a cardboard band 2cm wide with a diameter of 8cm as coil core. I wrapped around the band about 160 turns of 0.5mm Ø enameled copper wire. The inductance value approximation of the resulting coil can be calculated with the following formula:

$$L(uH) = \frac{0.8(r^2N^2)}{6r + 9l + 10d} \quad (1)$$

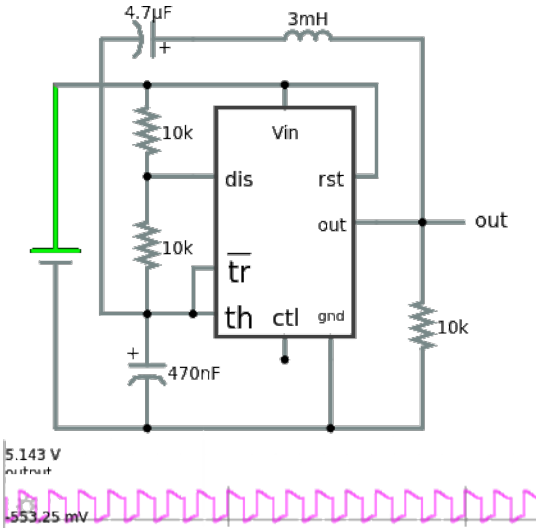


**Figure 2:** Metal detector circuit

Where $N$ is the number of turns, $r$ is the coil radius in inches, $l$ is the coil length in inches and $d$ is the winding depth in inches. For my coil the resulting inductance is about **3 mH**.

The *falstad.com* circuit simulation can be found at `/circuit/metal_detector.txt`

## 1.3. Arduino Nano

In order to read the frequency of pulses emitted by the metal detector circuit I chose an **Arduino Nano**. This tiny board is very cheap but quite powerful. I did not chose the Raspberry Pi, which I used for the robot, as a way to keep the metal detector an independent module which could also be used separately for different projects.

The Arduino Nano program was inspired by a *Nik Gammon* blog post about timers in ATMega328 chips.

ATMega328 chips have three timers. **Timer 0** is used to count approximately every millisecond since the current program is running and its value can be accessed through the *millis()* function. **Timer 1** and **2** can be used for different purposes like to generate a PWM output with the *analogWrite()* function. Timer 2 interrupts have an higher priority than timer 1 interrupts, which have an higher priority than timer 0 interrupts.

Timers 1 and 2 can also be used to count the number of events which cause a rising pulse on a digital pin during a specified interval. For example, if 10000 events occur in one second the frequency is 10kHz, but if the time interval would be 100ms and the events recorded 1000, the frequency would be the same.

Timer 1 is configured to count the number of times that a leading edge is detected on a specific pin. Each event increments the internal counter in the timer. When the 16-bit counter overflows (i.e. more than $2^{16} = 65536$ events are recorded) is triggered an overflow interrupt which counts the number of overflows occurred. When the time is up, the total number of counts is the number of overflows multiplied by 65536 plus the current counter contents of timer 1.

Timer 2 is used to count the user-defined time interval in which the total counts recorded by timer 1 occurred. Since clock speed in ATMega328 is 16MHz it runs with a period of $\frac{1}{16000000} = 62.5ns$. The clock is indeed prescaled (a way for the counter to skip a certain number of microcontroller clock ticks) of 128, so it will "tick" every $\frac{1}{(\frac{16000000}{128})} = 8us$. In order to have an interrupt every 1ms, timer 2 needs to count up to $\frac{1000us}{8us} = 125$. When timer 2 interrupt is triggered it is checked if the required number of milliseconds for the time interval is up. When this condition is reached the frequency of pulses can be calculated dividing the total count by the time interval.

The Arduino Nano code can be found at `code/arduino/freq_reader.ino`.

## 1.4. Results

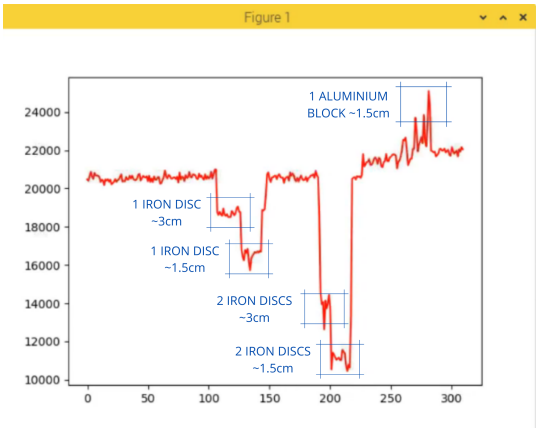


**Figure 3:** Metal detector frequencies when carried close to metal objects

The video `videos/metal_detector.mp4` shows how the metal detector frequency varies when carried close to metal objects of different sizes, at

different distances and of different types of metal.

**Figure 3** shows the same results demonstrated in the video mentioned above. When an iron disc is placed at about 3cm far from the metal detector the frequency drops of almost 2kHz, when the distance is 1.5cm the frequency drops of about 4kHz. The drop increases when the piece of metal is doubled (2 iron discs). When an aluminium power bank is carried about 1.5cm close to the metal detector, the frequency increases.

## 2. Exploratory Robot

The robot consists of a plexiglass plane (thickness 0.4cm), a breadboard, 4 DC motors (3-6V), a Raspberry Pi 3B+ and a PS2 EyeToy webcam. Originally I used the Raspberry Pi Camera Module but it stopped working all of a sudden and I did not have time to replace it.

### 2.1. Body design

The body design was first 3D modeled with *Blender*. The Blender file can be found at `model/body_design.blend`, the sizes are the actual cm values I used to build the robot body.

**Figure 4** shows the rendered Blender model. In the front of the body is placed the metal detector, while in the back the breadboard and the Raspberry Pi. In the middle there is a piece of plexiglass which holds the webcam.

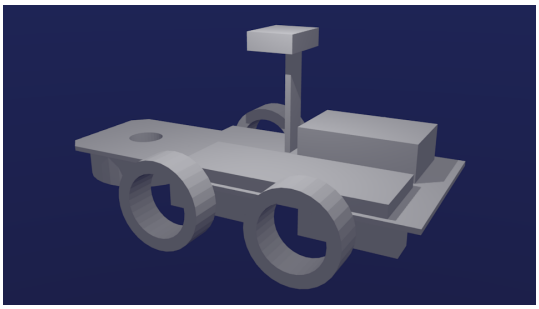

**Figure 4:** Robot body 3D model

### 2.2. Circuit design

The electronic circuit of the robot is straightforward, the four DC motors are connected to two L293D Motor Drivers which act as four H-bridges, indeed motors can move either forward or backward.

**Figure 5** is a detailed representation of the circuit. The outgoing wires from the L293D are connected to the Raspberry Pi. The complete Fritzing project can be found at `circuit/robot_circuit.fzz`.

For each motor the IC L293D has 3 pins: enable, input1 and input2. When the current flows through both enable and one input, the motor moves in one direction, when the current flows through the other input the motor moves in the opposite direction. Indeed to move forward the program sends current to the enable pins and to one input for each motor which should move forward. Similarly to mode backward, right and left.

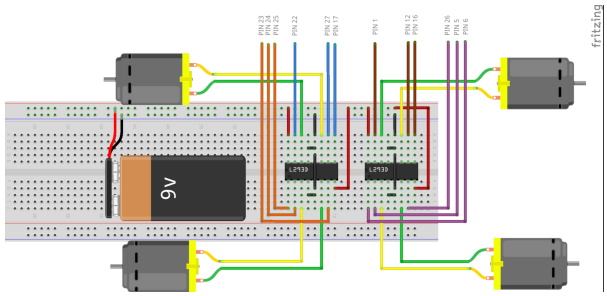


**Figure 5:** Exploratory robot circuit

### 2.3. Raspberry Pi

In the following sections I will describe how I setup an old PlayStation 2 camera to stream the live video of the exploratory robot and the idea behind the exploration algorithm.

#### 2.3.1. Camera

I initially mounted the Raspberry Pi Camera Module, which has easy to use Python APIs and did a great job until one day completely stopped working. Instead of removing the live video streaming feature from the robot I looked for my old PlayStation 2 webcam.

First the **v4l-utils** package is required. This package provides a series of utilities for media devices, allowing to handle the proprietary formats available at most webcams (libv4l), and providing tools to test V4L devices. In my Raspberry Pi was installed by default, otherwise it can be installed with the instruction:

```
sudo apt install v4l-utils
```

Second I installed the **motion** package. Which is a highly configurable program that monitors video signals from many types of cameras. It can be installed with the instruction:

```
sudo apt install motion
```

In order to remotely access the live video streaming some motion configuration parameters must be changed. It can be done by editing the following configuration file:

```
sudo nano /etc/motion/motion.conf
```

The following parameters must be set:

```
daemon on
stream_localhost off
width 640
height 480
```

Finally the motion service must be restarted for the changes to take place:

```
sudo service motion restart
```

With this setup the camera will stream the video on: `192.168.1.173:8081` (the IP could be different).

### 2.3.2. Exploration Algorithm

In this section I will describe how the exploration algorithm works, it is not complex but is adaptable to different exploration maps. The code can be found at `code/raspberrypi/exec.py`.

The ground is divided into an NxN grid, where N is the number of cells the map has for each side, each cell is about 20cm. The robot will move through recording frequencies from the metal detector for each cell and displaying a heatmap with these values.

The metal detector frequency is read through the serial port 115200 where the Arduino Nano sends recorded values. This task is done on a separate thread of the main process. Only the last 8 frequencies are kept, which is the amount of readings the Arduino can make while the robot moves forward of one cell in the map.

In order to show a live map of the frequencies recorded for each cell, a new process must be created. In this process a heatmap is periodically plotted with the `matplotlib` Python library and saved as PNG image.

To remotely access the heatmap plot a very simple webserver is setup in another process. It just returns an HTML file which shows the last saved heatmap plot image. This webserver is listening on `192.168.1.173:8082` (the IP could be different).

In **Figure 6** is shown an example of the heatmap of a square ground map with frequency values recorded from the metal detector (3 metal objects were detected).

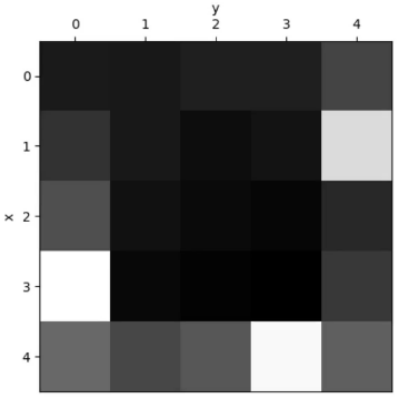


**Figure 6:** 5x5 heatmap example

In order to track the robot in the space, two vectors describe its location. One vector tracks its position in the whole map (**world space** coordinates), the other tracks its orientation, for instance if it is leading to North, West, etc. (**object space** coordinates).

When the robot moves to a new cell the algorithm scans all the cells around that one and checks if there is an unexplored cell. To do so it just rotates the orientation vector toward each cell, sums the obtained vector with the position vector and checks if the cell was not explored yet. To rotate a vector is used a simple rotation matrix as follows:

$$R_v = \begin{bmatrix} cos\ \theta & -sin\ \theta \\ sin\ \theta & cos\ \theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

However the grid is a discrete map, and using plain sine and cosine would skip corners of the square of cells. To avoid this problem I configured the sin and cos functions as shown in **Figure 7**. If no unexplored cell is found in the square of cells around the robot, this process is repeated with the cells at distance 2 from the robot. And so on until a cell is found or all the cells have been visited.

When an unexplored cell is found the robot rotates toward that cell and moves forward of the number of cells returned by the algorithm. When it reaches the desired place it rotates toward the closest angle multiple of $90°$. For instance, if the robot had to move of $157.5°$, after reaching the

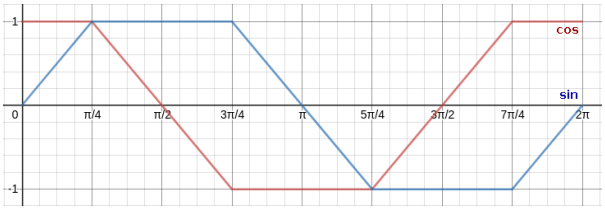


**Figure 7:** Sine and cosine functions

cell it will rotate $22.5°$ more in order to align with the $180°$. This will keep the orientation toward one of the four main cardinal directions.

### 2.4. Robot assembly

**Figure 8** is a picture of the assembled robot. The metal detector coil (in the front), the camera (in the middle), the Arduino Nano and the Raspberry Pi (in the back) are easily recognizable.

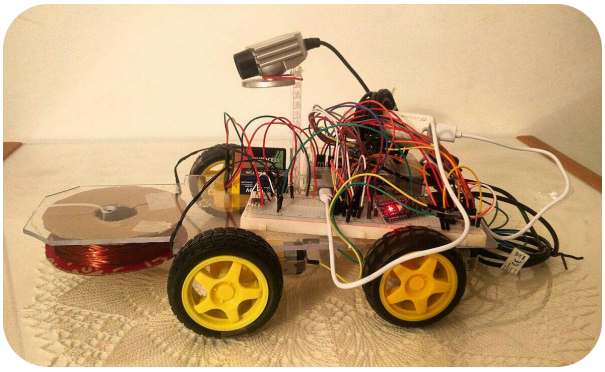

**Figure 8:** Assembled robot

## 3. Results

The file `videos/robot_test.mp4` shows a video example of the exploration of a square area of 5x5 cells (about $1m^2$). Three iron discs are positioned along the outer sides of the area. The live video streaming of the webcam is shown in the top left corner of the video. While in the top right corner are shown the live updates of the metal detector frequencies recorded in each explored cell.

During the exploration the heatmap shows light colors because the map is initialized to 1, while the frequencies are much higher values. When the exploration is complete and all values are in the same order of magnitude, the map is recolored with appropriate shades (as **Figure 6**).

**Figure 9** is a screenshot of the video described above.

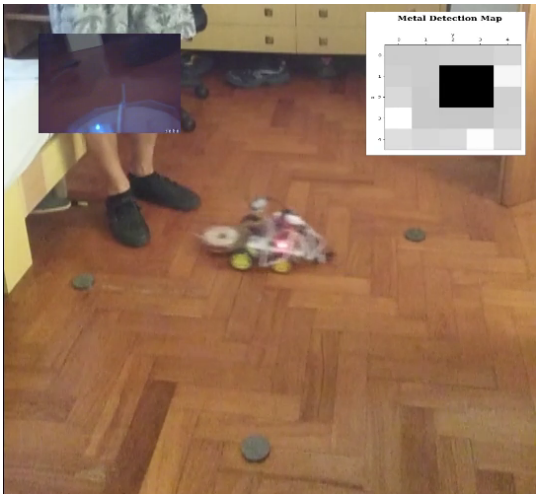


**Figure 9:** Screenshot from exploration test video

## 4. Conclusions

In this project I built a working metal detector robot which autonomously scans a ground map and looks for unexplored areas, providing users with a visual feedback.

Personally I strengthened my knowledge on electronic circuits such as make a custom coil and deal with capacitors and inductors. I have become familiar with very popular and useful integrated circuits. And learned in detail the features of the Arduino Nano timers and the Raspberry Pi board.

Possible **extensions** to the project could be improving accuracy and depth of the metal detector using a different design, or moving the metal detector with a servo (I could not implement this feature because I only had micro servos which could not carry its weight).