PicoPi Robot

november 2, 2021 · Mechatronica, Microcontroller, Raspberry Pi, Software

Auteurs

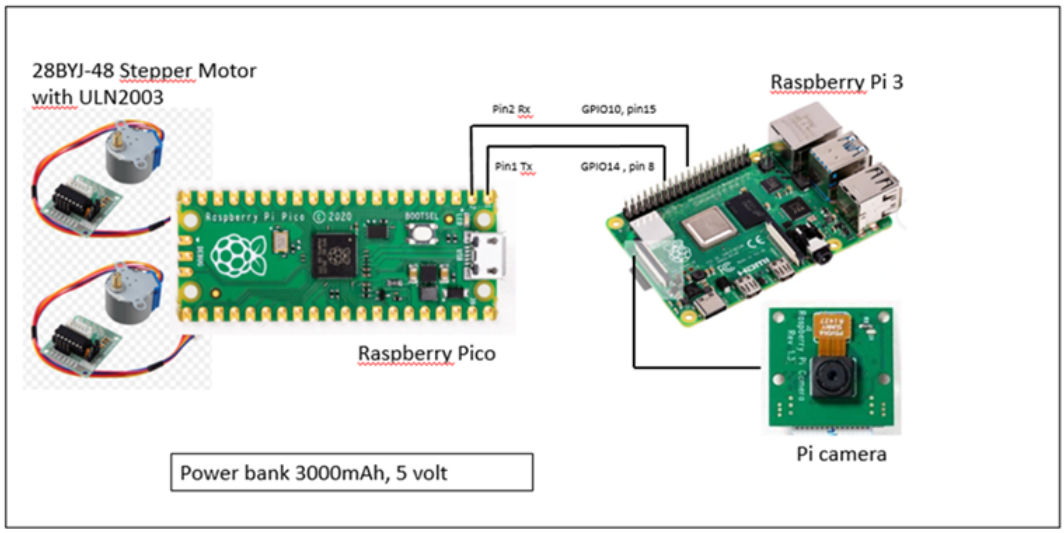
ZEH Otten

Line following with a camera

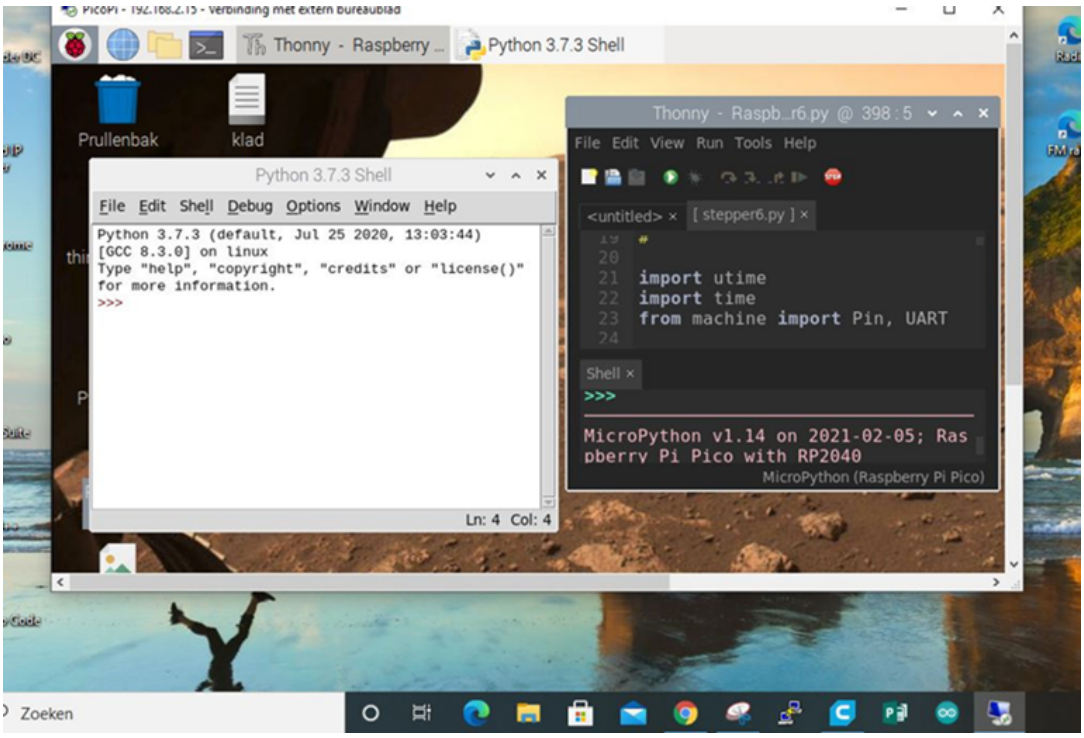
In this article I describe the construction of a robot that is able to follow a line with a camera. The project is not yet finished, but so far it has resulted in a working prototype.

Raspberry Pi (3) + Pico = PicoPi
The Pi3 is logged into the (home) WiFi network. The (linux) desktop is set up via the command install ***apt-get xrdp*** on the Pi3. Now the Pi3 can be taken over headless from a (windows) computer or laptop in the same WiFi network via a 'desktop session'. Programming, testing and ultimately line tracking can be managed completely wireless from the laptop. The video images can also be easily followed on the desktop via this connection. An ideal development and test environment.

The drive of my robot consists of two stepper motors. The Pi3 could control the two steppers. However, out of interest in innovation and challenge, I have chosen to control the stepper motors with a new Raspberry Pico microcontroller, based on the RP2040 chip. Control commands for the stepper motors are sent to the Pico via a serial connection between the Pi3 and the Pico. The control commands are 'devised' by the Pi3.
The power supply of 5 volts is provided by a phone power bank of 3000mAh. This feeds the Pi3 and the Pico and supplies enough to keep the robot running for several hours. Schematically it looks something like this:



The picture below gives an impression of how I work on my Windows 10 laptop, with in the foreground the desktop session of the Pi3 with the open Python IDE and the Thonny programming environment of the Pico.



The Raspberry Pi Pico is programmable in C and MicroPython languages. I chose the latter.
When you install the latest version of Thonny (a Python IDE for beginners) on the Pi3, it is easy to program the Pico via the USB port of the Pi3. There is also a Windows version of Thonny so that you can also program the Pico via Windows 10 and a USB port.

Via a second serial connection, the Pi3 communicates with the Pico which receives control commands. For example, I formulated commands to make the two stepper motors make a defined movement. For example: ‘MF100’ (move forward 100 steps) or ‘TL100’ (turn left 100 steps).
The serial connection is easily realized with the following micro-python code on the Pico:

```
    run = True
    while run:
        if uart.any():
            received_data = uart.readline()
            s=received_data.decode()
            #print(s, len(s))
            if len(s) == 6 :
                run = False

    utime.sleep(0.01)
    return
```

Then the stepper motors are controlled with the following micro-python code:

```
def step_motorR(count):
    # count is het aantal stappen rechtsom
    # -count is het aantal stappen linksom
    global current_stepR
    while (count != 0):
        if (count < 0):
            high_motorR = stepRR[current_stepR]
        else:
            high_motorR = stepR[current_stepR]
        set_motorR_low(motorR)
        set_motorR_high(high_motorR)
        current_stepR += 1
        if current_stepR == len(stepR):
            current_stepR = 0
        time.sleep(speed)
        if (count < 0):
            count +=1
        else:
            count -=1
    return
```

All that remains now to be done is put together the correct motor commands to make PicoPi follow a line!

Image analysis
The Python program PicoPy.py runs on the Pi3. I will explain broadly how I arrived at this result.
The camera is a Pi3 camera that is widely used in Raspberry Pi projects. The camera is set to 320×240 pixels and a frame rate of 32 frames per second. For a simple line

```
#==========movement=================
def turnL(steps):
    for x in range(steps):
        step_motorL(1)
        step_motorR(-1)
    return

def turnR(steps):
    for x in range(steps):
        step_motorL(-1)
        step_motorR(1)
    return

def forward(steps):
```

detection this should be sufficient. The rawCapture variable contains the video frame to be edited:

```
yy  = 240  # x-as
xx  = 320  # y-as
# initialize parameters
camera            = PiCamera()
camera.resolution = (xx, yy)
camera.framerate  = 32
rawCapture        = PiRGBArray(camera, size=(xx, yy))
```

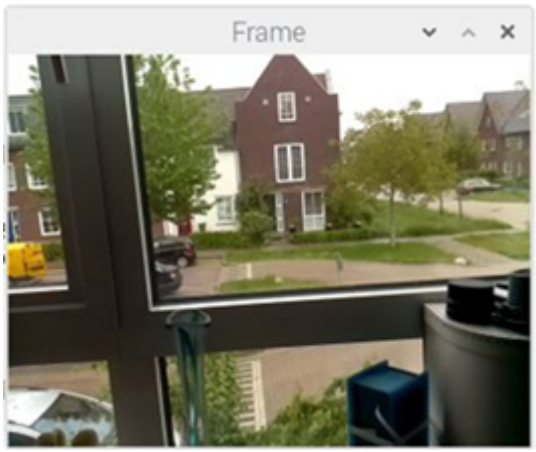
The main loop of the program is simple; determine the position of the line to follow for each frame from the rawCapture and then stay on the road!

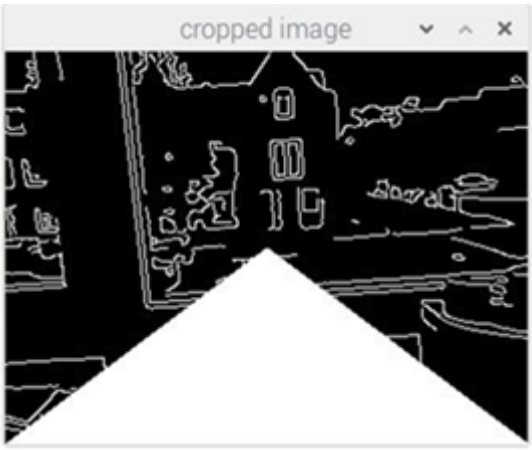
```
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):

    image = frame.array   # maak beeld van de omgeving
    getLane(image,1)      # bepaal de lijn en toon beeld
    controlLane(setpunt,afwijking,param)  # blijf op de weg !
```

Now OpenCV needs to get to work.
In the getLane() procedure, a grayscale image is created from each frame. The image is then ‘blurred’ and the information from the image is outlined via the Canny function.
Because we will soon be only interested in a line and not so much in the immediate vicinity, we can limit our image to a part of the frame, namely to the region of interest. This image is stored in the cropped_image:

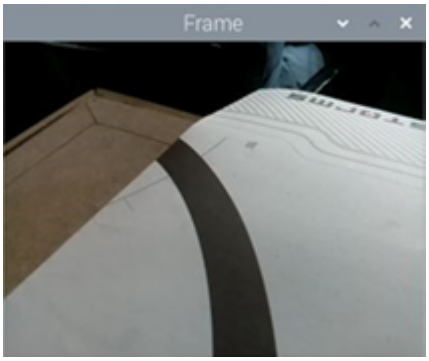
```
    gray_image    = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY )
    blur_image    = cv2.GaussianBlur(gray_image,(5,5),0)
    canny_image   = cv2.Canny(blur_image,100,200)
    cropped_image = region_of_interest(canny_image,
                    np.array([region_of_interest_vertices],np.int32),)
```

The result of processing the images with OpenCV is highly dependent on the amount of ambient light. Also , there are many parameters that must be specified when calling the OpenCV procedures . Nevertheless, a result can quickly be achieved as shown below:





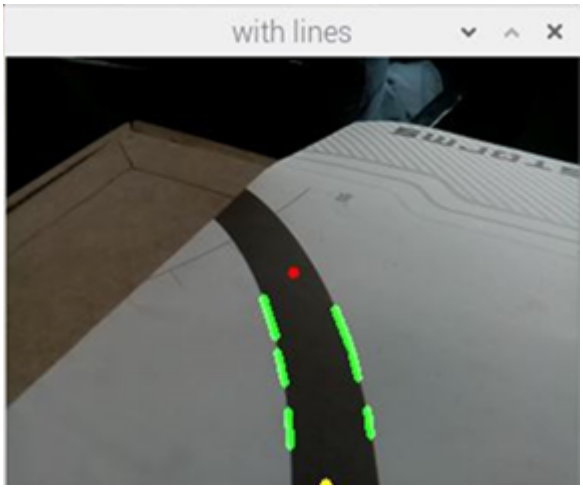
This result shows an example of a video frame showing the outlines of the frame after the blur and after the Canny function. The white triangle is the area chosen as the region of interest. In case we are going to follow a line and if we point the camera at a piece of 'line follow lane' then the following can be made visible with the above getLane() procedure:



Now comes the most important part of the whole line tracking program. The Hough filter.
This arithmetic filter is able to derive lines, circles and ellipses from a processed video frame.
Calling the Hough filter with the cropped_image goes like this:

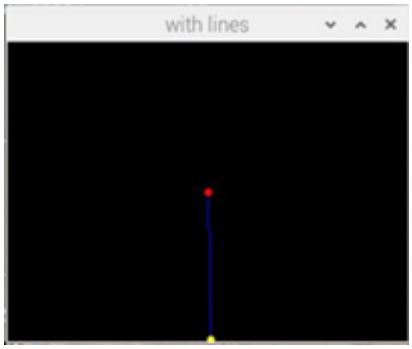
```
228     # zoek lijnen
229     # (default 6,pi/60, 160,100,10 )
230     # werkend 6,pi/60, 160,15,10 )
231     lines = cv2.HoughLinesP(cropped_image,
232                             6,
233                             np.pi/60,
234                             threshold = 10,
235                             minLineLength=15,
236                             maxLineGap=10)
237
238     if lines is not None:
```

As you can see it is just one call but with many parameters. I had to spend a lot of time (experimentally) determining some parameters to finally determine a few lines from the frame. In the picture below, the lines found by the Hough filter are shown in green in the region of interest in the original frame:

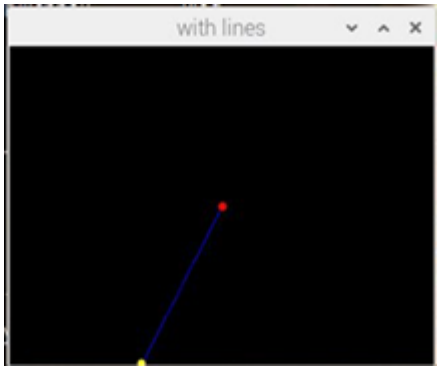


The Hough filter thus finds the left and right margins of the black line to be followed.
We would like to have the lines as shown in the image in an arithmetic form such as y=ax+b.
These lines are easy to deduce when you consider that the frame consists of an x,y coordinate system of 320×240 pixels.

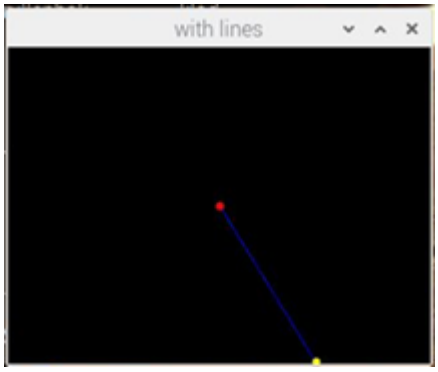
I now calculate an average line that runs between the green lines (blue line).
The mean line passes through the red (middle) point and intersects the x-axis at the yellow point. The yellow point will therefore change per image frame and shift across the x-axis, depending on the deviation of the robot from the line to be followed.
I am using this yellow point (x value, y=0) as my motor control set point.



eviation = almost 0: little control action; keep driving straight.



Deviation from the center: steer the robot to the right



Deviation from the center: steer the robot to the left

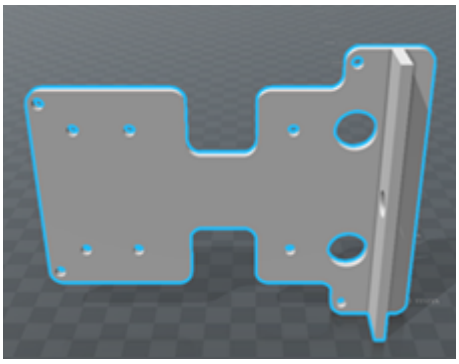
This way of controlling means that the robot must be positioned exactly in the middle of the line to be followed at the start of the line-following session.

The procedure controlLane() controls this process using a PID controller. Setting the PID parameters is a challenge in itself. In the end I have results with mainly P- and D action (and I-action on zero). The output of the controller is then a measure of the number of steps and direction for the control commands:

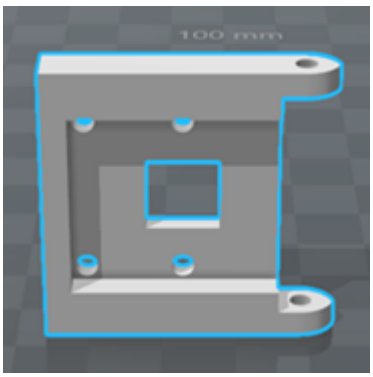
```
#-------------------------------------
def controlLane(setp, meting, parameters):
    Kp, Ki, Kd, Pb = parameters
    global prevT
    global eintegral
    global dedt
    global eprev

    currT = round(time.monotonic() * 1000)
    deltaT = (currT - prevT) # ordegrootte = 300 ms
    prevT = currT
    error = meting - setp
    dedt = (error - eprev) / deltaT
    eintegral = eintegral + error * deltaT
    eprev = error

    out = Kp * error +  Ki * eintegral + Kd * dedt

```

The robot PicoPi

Now everything should come together. The frame is now reused. Together with some parts, printed with a 3D printer, everything was built and put together, resulting in the following prototype ‘showpiece’ on the next page:



Main frame for PicoPi



Camera holder

Conclusion

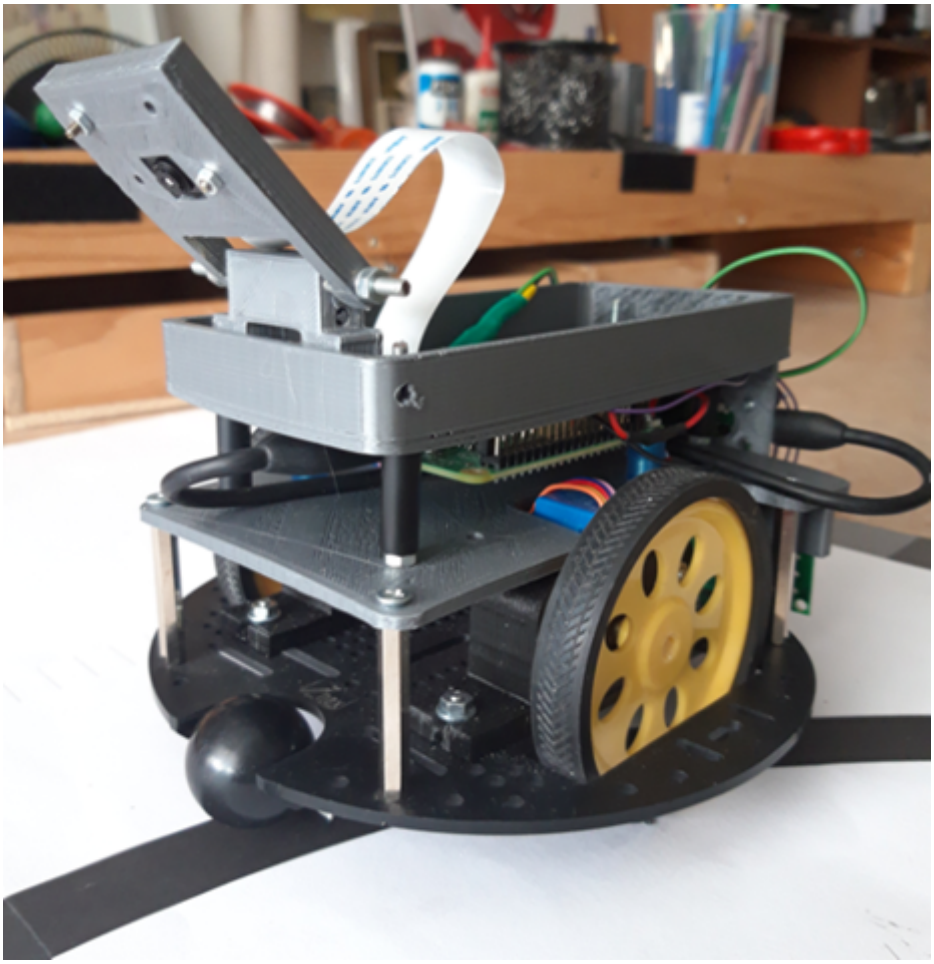
The line tracking based on a camera and OpenCV with a Pi3 and Pico is possible.
Serial communication between a Raspberry and a Pico is easy to realize. However, the programming of the uart should be better in terms of performance. The buffer was not emptied properly (probably due to limited programming experience). New data is now added to data that is still in the buffer. That’s why I applied some strange (auxiliary) constructs in the pico/micro-python software.

When using a Raspberry Pi4 (the powerful successor of the Pi3) it is possible to process higher resolution images. Whether this leads to better results will have to be investigated. The region of interest can also be expanded.

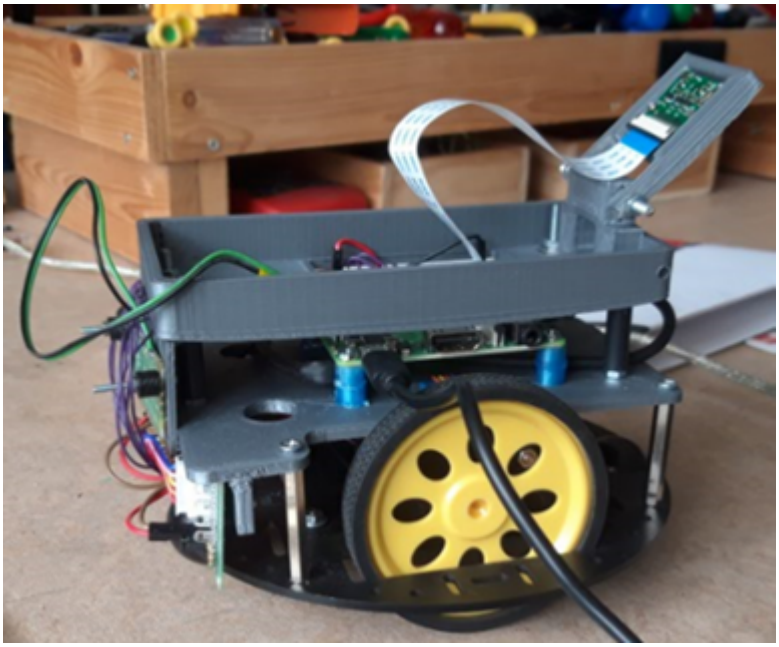
The prototype shown here works but there are still improvements to be made:

- The stepper motors used are weak and run very slowly. The robot is (much) too slow.
- Is the choice of the set point ideal?

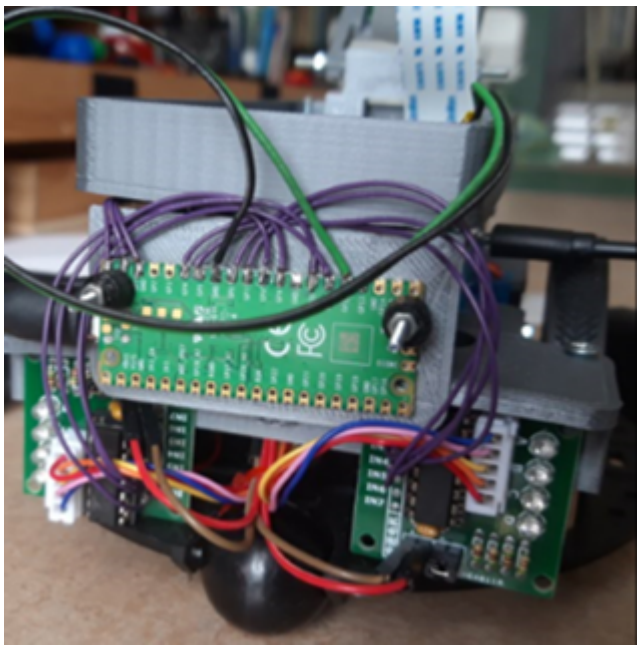
- Printing parts for the robot is excellent with a 3d printer, but then spend time on the design of the parts of the robot before you actually start building.
  My picoPi came about through an ‘organic growth model’; building with advancing insight..
  So if you look closely you will see an extra drilled hole or something that is not completely straight!



PicoPi finding lanes



There is room for the power bank



Pico controlling indirect the steppers

Share this:

Press this Twitter Facebook

Personaliseringsknoppen

Herbloggen Like

Wees de eerste die dit leuk vindt.

Gerelateerd

Intelligent mousetrap
5 oktober 2020
In "Mechatronica"

Aladdin: Stormlamp zonder olie.
15 juli 2021
In "Electronica"

Raspberry Pi (1): Eerste kennismaking
28 augustus 2013
In "Electronica"