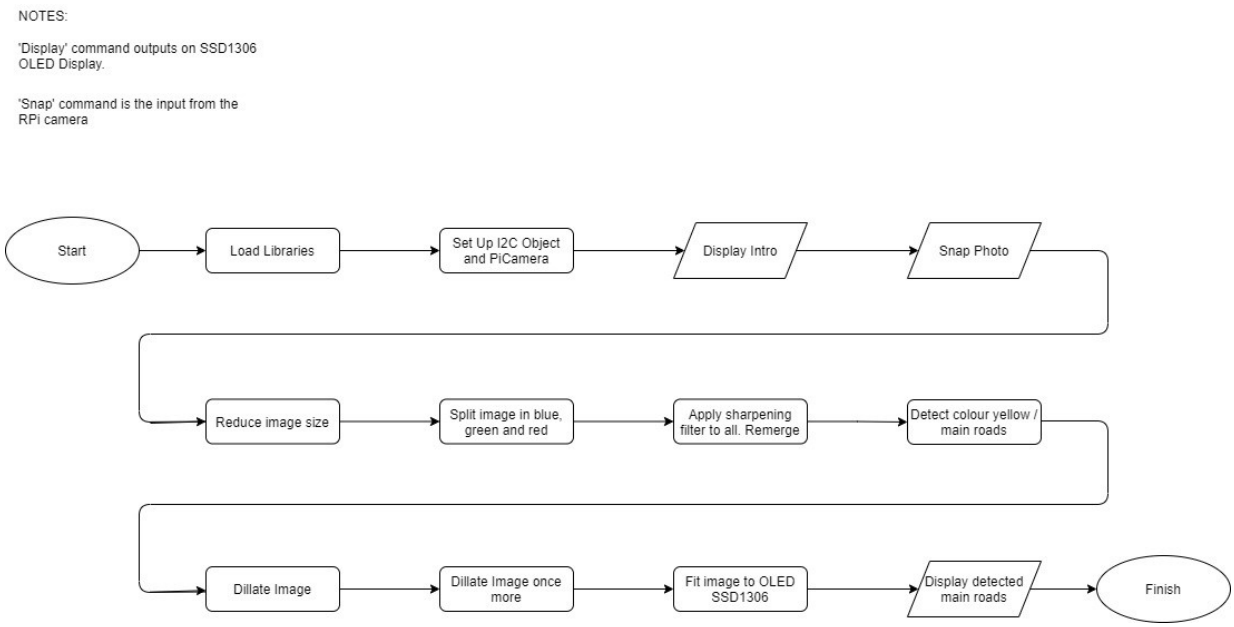
Saxion University of Applied Sciences
Smart Embedded Systems Minor
Vision – Spring 2020 / 2021
Luciano Regis Orlandini – 460952 – Applied Computer Science

Final Assignment

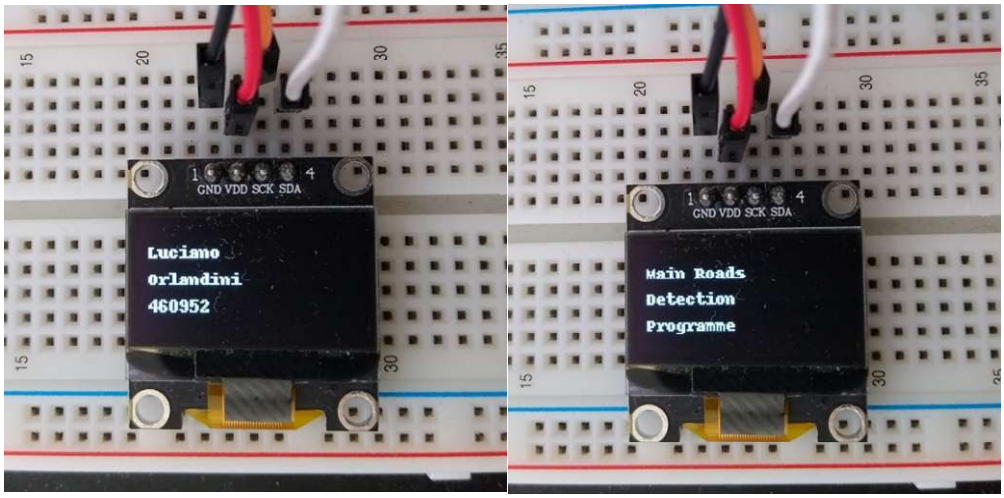
For this assignment I've created a small programme that does the following:

- Takes a photograph of a map using the RPi camera. This map can be printed or on screen.
- Sharpens the image so the colours become more contrasting
- Identifies the colour yellow which represents main roads in maps
- Creates a binary image where the pixels detected as yellow become white and all the rest black
- It dilates the image to remove most gaps which are caused by the street name which is printed on top of the road
- Resizes the image so it fits on the SSD1306 128 x 64 display, which is connected to the RPi via I2C interface.

A flowchart of this programme is below:



The programme starts by displaying on the screen a small intro showing my name and the programme title.

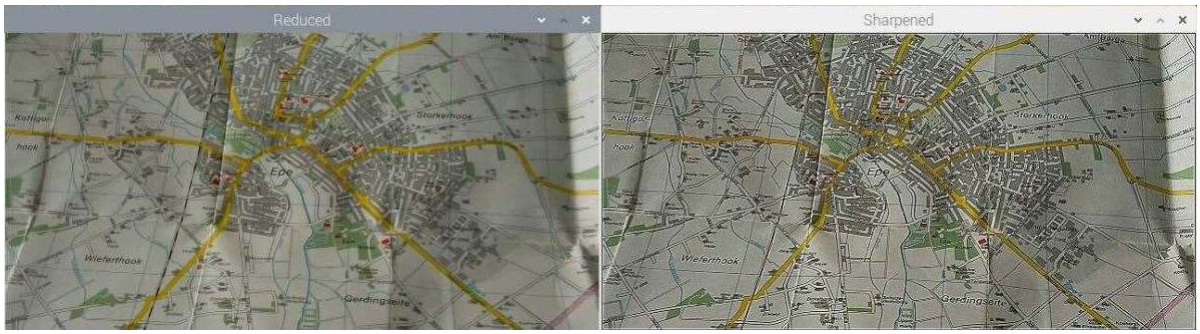


Then we first let the RPi camera take a photo in bgr (blue, green and red) of a printed map of Gronau, where I live.

*Photo from my phone to showcase full size map

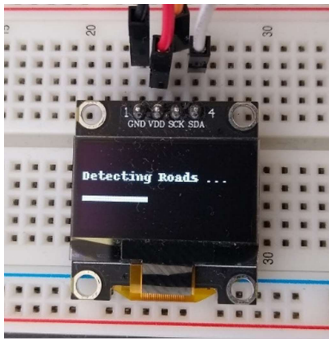
* Photo taken by Raspberry Pi

Then the image gets reduced in size and sharpened. In order to sharpen it using my existing functions from previous assignments, I had to first split the image into their 3 different colour channels, apply the sharpening filter to each of them and then remerge them. The images below show the ‘before and after’ of the sharpening.

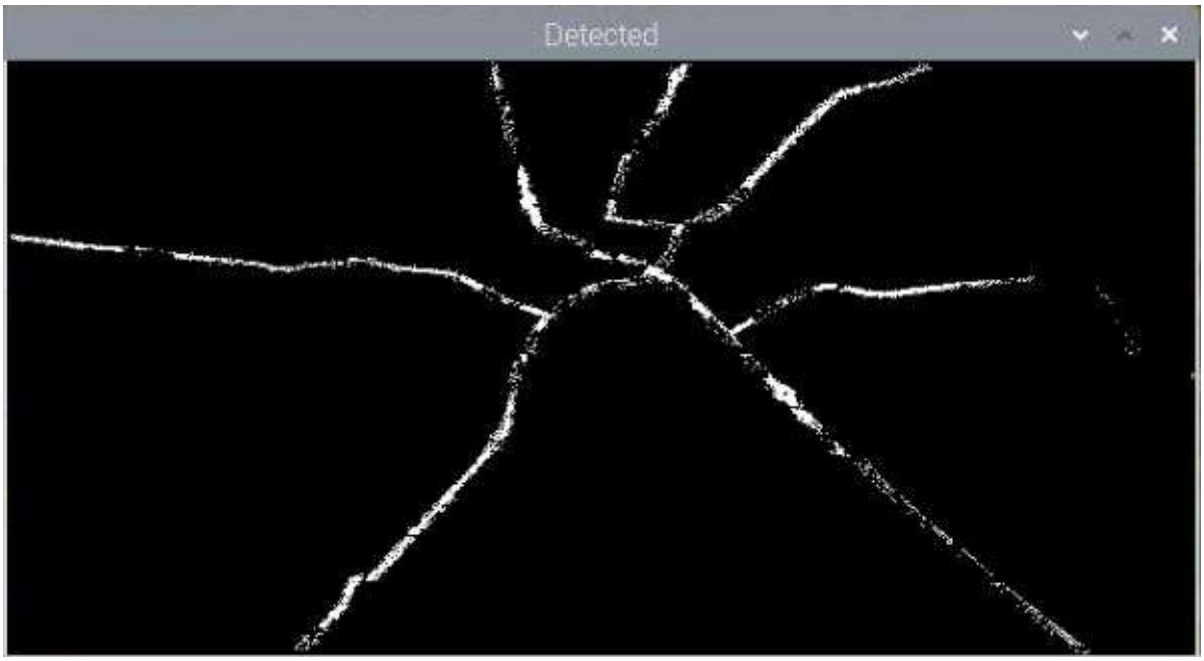


The sharper image helps with the accuracy of the colour detection.

During the image processing phase, a loading bar can be seen on the small display to showcase the progress.

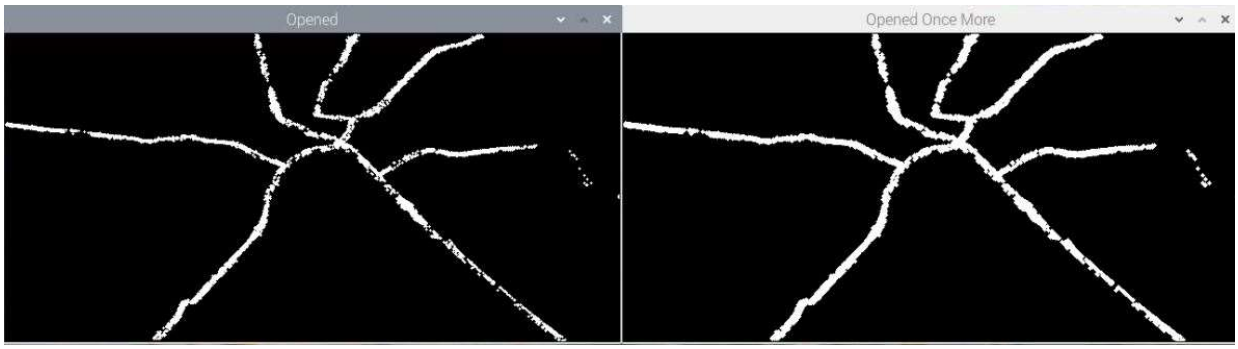


Afterwards the programme takes the sharper image and searches the colour yellow within a specified range. It then creates a binary image where the pixels detected as yellow become white and all the rest black.



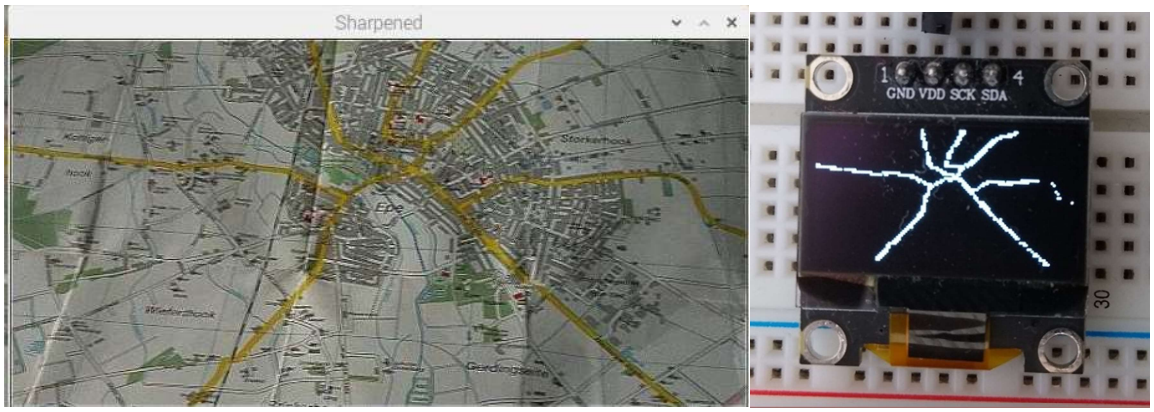
The roads shown in the image above have many black artefacts within as these were just noise or the street names which are printed in black on top of the roads, hence undetected.

To complete these spaces and create a higher definition road layout the programme dilates the image twice.



It is visible that after each dilation the roads layout is ‘fuller’.

Finally, the programme resizes this image once more so it will fit in the small OLED screen and displays it.



```
# Saxion University of Applied Sciences
# Smart Embedded Systems - Fall 2020/2021
# Vision
#
# Luciano Regis Orlandini - 460952
# Applied Computer Science
#
# Final Vision Assignment
#
# This programme will take photographs of a printed map which has
# main roads in colour yellow. The programme will process the image
# to filter out any smaller roads and will display the main ones
# on a SSD1306 display.

# Libraries required for OLED display from AdaFruit and to connect via I2C
from lib_oled96 import ssd1306
from smbus import SMBus
i2cbus = SMBus(1)

# Libraries required to take photos and image processing
import sys
sys.path.append('/usr/lib/python3/dist-packages/')
from picamera.array import PiRGBArray
from picamera import PiCamera
from PIL import Image
import time
import cv2
import numpy as np

# Initilise screen
oled = ssd1306(i2cbus) # Screen object
oled.cls() # Clears oled memory and screen

# Initialise RPi camera
camera = PiCamera()
camera.resolution = (1920, 1088)
rawCapture = PiRGBArray(camera)
time.sleep(1) # ALlows time for the sensor to warm up

# Sharpening filter kernel
sharpen =  np.array([[ 0,-1, 0],
                     [-1, 5,-1],
                     [ 0,-1, 0]])

# Loads image from folder
def get_image():
    return cv2.imread("map.jpg")

# Introduction showin my name for 5 seconds
def intro():
    oled.canvas.text((5,12), 'Luciano\nOrlandini\n460952', fill=1)
    oled.display()
    time.sleep(3)
    oled.cls()
    oled.canvas.text((5,12), 'Main Roads\nDetection\nProgramme', fill=1)
    oled.display()
    time.sleep(3)
    oled.cls()
```

```
# Takes photo using video port as photo port is defective on my device
def snap_photo():
    camera.capture(rawCapture, format="bgr", use_video_port=True)
    pic = rawCapture.array
    cv2.imshow('Image', pic)
    cv2.waitKey(0)
    oled.canvas.text((0,18), 'Detecting Roads ...', fill=1)
    oled.canvas.line([(0,40), (1,40)], fill = 1, width = 4)
    oled.display()
    cv2.imwrite('map.jpg', pic)
    reduced = reduce_size(get_image())
    b,g,r = cv2.split(reduced)
    b = apply_filter(b, sharpen)
    oled.canvas.line([(0,40), (20,40)], fill = 1, width = 4)
    oled.display()
    g = apply_filter(g, sharpen)
    oled.canvas.line([(0,40), (35,40)], fill = 1, width = 4)
    oled.display()
    r = apply_filter(r, sharpen)
    oled.canvas.line([(0,40), (50,40)], fill = 1, width = 4)
    oled.display()
    sharp = cv2.merge((b,g,r))
    cv2.imwrite('map.jpg', sharp)

# Applies filter to snapped photo
def apply_filter(image, kernel):
    i = image
    k = kernel
    rows = len(i)
    columns = len(i[0])
    proc_i = np.zeros((rows, columns), np.int16)

    # Applies black border around image
    for x in range (rows):
        if x == 0 or x == (rows - 1):
            for y in range (columns):
                i.itemset((x,y), 0)
        else:
            i.itemset((x,0), 0)
            i.itemset((x,(columns -1)), 0)
    # Passes kernel through image and convolutes pixel values to anchor
    kx = 1
    ky = 1
    for x in range (1, rows - 1):
        for y in range (1, columns - 1):
            proc_i.itemset((x,y), ((k.item(kx-1, ky-1) * i.item(x-1, y-1)) + \
                                   (k.item(kx-1, ky) * i.item(x-1, y)) + \
                                   (k.item(kx-1, ky+1) * i.item(x-1, y+1)) + \
                                   (k.item(kx, ky-1) * i.item(x, y-1)) + \
                                   (k.item(kx, ky+1) * i.item(x, y+1)) + \
                                   (k.item(kx+1, ky-1) * i.item(x+1, y-1)) + \
                                   (k.item(kx+1, ky) * i.item(x+1, y)) + \
                                   (k.item(kx+1, ky+1) * i.item(x+1, y+1)) + \
                                   (k.item(kx, ky) * i.item(x, y))))
```

```
    for x in range(1, rows - 1):
        for y in range(1, columns - 1):
            if proc_i.item(x,y) < 0:
                i.itemset((x,y), 0)
            elif proc_i.item(x,y) > 255:
                i.itemset((x,y), 255)
            else:
                i.itemset((x,y), np.uint8(proc_i.item(x,y)))

    return i

# Detecs colour yellow which are the main roads on maps
def detect_yellow():
    roads = get_image()
    cv2.imshow("Sharpened", roads)
    cv2.waitKey(0)
    # Define yellow lower and upper limits
    lw = np.array([ 0,105,105], dtype = "uint8")
    up = np.array([80,255,255], dtype = "uint8")
    # Find the colors within the specified boundaries and make it binary
    binary = cv2.inRange(roads, lw, up)
    oled.canvas.line([(0,40), (70,40)], fill = 1, width = 4)
    oled.display()
    cv2.imshow("Detected", binary)
    cv2.waitKey(0)

    return binary

# Erodes or Dilates image, used for removing noise and opening white pixels
# erosion parameter set to True of false to signal operation required
def ero4_dil4(image, name, erosion):
    i = image
    rows = len(i)
    columns = len(i[0])
    proc_i = np.zeros((rows, columns), np.uint8)
    k = 255

    if erosion:
        k = 0

    # Runs kernel through image and erodes
    for x in range (1, rows - 1):
        for y in range (1, columns - 1):
            if i.item(x,y) != k:
                condition_met = (i.item(x,y) != i.item(x-1, y))   or \
                                (i.item(x,y) != i.item(x, y-1))   or \
                                (i.item(x,y) != i.item(x, y+1))   or \
                                (i.item(x,y) != i.item(x+1, y))
                if condition_met:
                    proc_i.itemset((x,y), k)
                else:
                    proc_i.itemset((x,y), i.item(x,y))
            else:
                proc_i.itemset((x,y), i.item(x,y))
```

```
    # Displays and returns filtered image
    cv2.imshow(name, proc_i)
    cv2.waitKey(0)
    oled.canvas.line([(0,40), (90,40)], fill = 1, width = 4)
    oled.display()

    return proc_i

# Reduces image partially for faster processing
def reduce_size(image):
    #reduced = cv2.resize(image, (384, 192))
    reduced = cv2.resize(image, (640, 320))
    #reduced = cv2.resize(image, (896, 448))
    cv2.imshow("Reduced", reduced)
    cv2.waitKey(0)

    return reduced

# Resizes image to 128 x 64 so it fits on SSD1306 display
def fit_to_screen(image):
    reduced = cv2.resize(image, (128, 64))
    cv2.imshow("Fitted to OLED", reduced)
    cv2.waitKey(0)
    oled.canvas.line([(0,40), (110,40)], fill = 1, width = 4)
    oled.display()
    cv2.imwrite("Reduced.jpg", reduced)

# Displays image on SSD1306 display
def display_image():
    oled.cls()
    oled.canvas.bitmap((0,0), Image.open("Reduced.jpg"), fill = 1)
    oled.display()
    time.sleep(5)

# Main
intro()
snap_photo()
main_rds = detect_yellow()
opened = ero4_dil4(main_rds, "Opened", False)
opened2 = ero4_dil4(opened, "Opened Once More", False)
fit_to_screen(opened2)
display_image()
cv2.destroyAllWindows()
```