In [1]:
```
from pynq import Overlay, MMIO, lib
from pynq.lib.video import VideoMode
from PIL import Image
import cffi
from time import sleep
import os
import numpy as np
os.environ["OPENCV_LOG_LEVEL"]="SILENT"
# initialize camera from OpenCV
import cv2
```

## Before starting this piece of code be sure that SW0 on board is in OFF position

In [2]:
```
overlay = Overlay("design_1_wrapper.xsa")
```

Class to manage the convolution filter mapped on FPGA. It provides method to modify "on fly" the kernel (7x7)

In [3]:

```
class Convolution_Filter:
    def __init__(self, overlay, base_address=0x43C10000, address_range=0x10000, address_offset=0x40):
        self.base_address = base_address
        self.address_range = address_range
        self.address_offset = address_offset
        self.offset = 0x04
        self.mmio = MMIO(base_address, address_range)
        self.conv = overlay.filter.convolution_filter

    def update_filter(self, fil):
        if(len(fil) != 51):
            print("La lunghezza del filtro deve essere di 51 elementi")

        address = self.address_offset
        data = 0x00000000
        bits_shift = 0
        counter = 0

        for el in fil:
            if(bits_shift >= 32):
                self.mmio.write(address, data)
                data = 0x00000000
                bits_shift = 0
                address = address + self.offset

            counter += 1
            data = data | (el << bits_shift)
            bits_shift += 8
            if(counter >= 51):
                self.mmio.write(address, data)

    def print_filter(self):
        f1 = self.conv.mmio.array.view('int8')[0x40:0x71]
        f2 = self.conv.mmio.array.view('int8')[0x71:0x73]

        print(f1.reshape((7,7)))
        print(f2.reshape((1,2)))
```

Class to manage OV7670 sensor. It provides basic methods to write and read sensor's registers and a basic setup that works quite well in our configuration

In [4]:

```
class OV7670:
    def __init__(self, iic):
        self.OV7670_SLAVE_ADDRESS = 0x21

        _ffi = cffi.FFI()
        self.tx_buf = _ffi.new("unsigned char [32]")
        self.rx_buf = _ffi.new("unsigned char [32]")

        self.iic = iic

    def write_register(self, reg, data):
        self.tx_buf[0] = reg
        self.tx_buf[1] = data

        self.iic.send(self.OV7670_SLAVE_ADDRESS, self.tx_buf, 2, 0)

    def read_register(self, reg):
        self.tx_buf[0] = reg

        self.iic.send(self.OV7670_SLAVE_ADDRESS, self.tx_buf, 1, 0)
        self.iic.receive(self.OV7670_SLAVE_ADDRESS, self.rx_buf, 1, 0)

        return self.rx_buf[0]

    def default_setup(self):
        self.write_register(0x12, 0x80)
        sleep(1)
        self.write_register(0x0E, 0x01)
        self.write_register(0x0F, 0x4B)
        self.write_register(0x16, 0x02)
        self.write_register(0x1E, 0x07)
        self.write_register(0x21, 0x02)
        self.write_register(0x22, 0x91)
        self.write_register(0x29, 0x07)
        self.write_register(0x33, 0x0B)
        self.write_register(0x35, 0x0B)
        self.write_register(0x37, 0x1D)
        self.write_register(0x38, 0x01)
        self.write_register(0x0C, 0x00)
        self.write_register(0x3C, 0x78)
        self.write_register(0x4D, 0x40)
        self.write_register(0x4E, 0x20)
        self.write_register(0x74, 0x10)
```

```
        self.write_register(0x8D, 0x4F)
        self.write_register(0x8E, 0x00)
        self.write_register(0x8F, 0x00)
        self.write_register(0x90, 0x00)
        self.write_register(0x91, 0x00)
        self.write_register(0x96, 0x00)
        self.write_register(0x9A, 0x00)
        self.write_register(0xB0, 0x84)
        self.write_register(0xB1, 0x04)
        self.write_register(0xB2, 0x0E)
        self.write_register(0xB3, 0x82)
        self.write_register(0xB8, 0x0A)
```

Usage example of OV7670 class to program sensor with a basic setup

In [5]:
```
iic = overlay.axi_iic
ov7670 = OV7670(iic)
ov7670.default_setup()
```

# Before exexuting this piece of code set SW0 on

Usage example of convolution filter class

In [6]:

```
sharpen_filter = [
    1, 0, 0, 0, 0, 0, 0,
    0, 1, 0, 0, 0, 0, 0,
    0, 0, 1, 0, 0, 0, 0,
    0, 0, 0, 1, 0, 0, 0,
    0, 0, 0, 0, 1, 0, 0,
    0, 0, 0, 0, 0, 1, 0,
    0, 0, 0, 0, 0, 0, 1,
    7, 0]

neutral_filter = [
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    1, 0]

vertical_filter = [
    -1, -2, -4, 0, 4, 2, 1,
    -1, -2, -4, 0, 4, 2, 1,
    -2, -4, -6, 0, 6, 4, 2,
    -4, -6, -8, 0, 8, 6, 4,
    -2, -4, -6, 0, 6, 4, 2,
    -1, -2, -4, 0, 4, 2, 1,
    -1, -2, -4, 0, 4, 2, 1,
    120, 127]

fil = Convolution_Filter(overlay)
fil.update_filter(sharpen_filter)
```

In [7]:
```
# Configuration of vdma with a resolution of 800x600 and 24 bit for each pixel
vdma = overlay.VDMA.axi_vdma

vdma.readchannel.reset()
vdma.readchannel.mode = VideoMode(800, 600, 24)
vdma.readchannel.start()

vdma.writechannel.reset()
vdma.writechannel.mode = VideoMode(800, 600, 24)
vdma.writechannel.start()

frame = vdma.readchannel.readframe() # Needed because first frame is always black

vdma.readchannel.tie(vdma.writechannel) # Connect input directly to output of vdma
```

In [8]:
```
frame = vdma.readchannel.readframe()


#img = cv2.GaussianBlur(frame,(5,5),0)
#edge = cv2.Canny(frame, 100, 200)

#edge = cv2.Canny(img, 50, 150)
#edge_rgb = cv2.cvtColor(edge, cv2.COLOR_GRAY2RGB)
#img2 = Image.fromarray(edge_rgb, 'RGB')
#display(img2)
```

In [9]:

```
def detect_lane_markings(image):
    lane_check = 0
    # Convert image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur to smooth the image
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    # Perform edge detection using Canny
    edges = cv2.Canny(blurred, 50, 150)

    # Define region of interest (top part of the image with a limited distance from the top)
    height, width = edges.shape
    mask = np.zeros_like(edges)

    # Define the distance from the top of the image to disregard lane markings
    distance_from_top = 300  # Adjust this value as needed

    polygon = np.array([[(0, distance_from_top), (width, distance_from_top),
                         (width, height), (0, height)]], np.int32)
    cv2.fillPoly(mask, polygon, 255)
    masked_edges = cv2.bitwise_and(edges, mask)

    # Detect lines using Hough transform
    lines = cv2.HoughLinesP(masked_edges, rho=1, theta=np.pi/180, threshold=15, minLineLength=10, maxLine

    # Create a blank image with the same size as the original image
    lane_markings_image = np.zeros_like(image)

    # Draw detected lines on the top part of the image
    if lines is not None:
        lane_check = 1
        for line in lines:
            x1, y1, x2, y2 = line[0]
            cv2.line(lane_markings_image, (x1, y1), (x2, y2), (0, 255, 0), 5)

    return lane_check, lane_markings_image
```

In [10]:

```
def detect_stop_sign(image):
    stop_check=0
    # Convert image to grayscale
    #gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur to reduce noise
    blurred = cv2.GaussianBlur(image, (5, 5), 0)

    # Detect edges using Canny
    edges = cv2.Canny(blurred, 30, 100)

    # Find contours in the edge-detected image
    contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Iterate through contours to find potential stop signs
    stop_sign_image = np.zeros_like(image)
    for contour in contours:
        # Approximate the contour to a polygon
        approx = cv2.approxPolyDP(contour, 0.03 * cv2.arcLength(contour, True), True)

        # Check if the polygon has 8 vertices (indicating a stop sign)
        if len(approx) == 8:
            # Calculate the bounding box for the polygon
            (x, y, w, h) = cv2.boundingRect(approx)

            # Calculate the aspect ratio of the bounding box
            aspect_ratio = w / float(h)

            # Check if the aspect ratio is approximately 1 (indicating a square)
            if 0.8 <= aspect_ratio <= 1.2:
                # Check if the contour area is within a reasonable range
                contour_area = cv2.contourArea(contour)
                if 2000 <= contour_area <= 5000:  # Adjust area thresholds as needed
                    # Draw a green rectangle around the detected stop sign
                    cv2.rectangle(stop_sign_image, (x, y), (x + w, y + h), (0, 255, 0), 3)
                    stop_check=1

    return stop_sign_image
```

In [ ]:

```
```

In [11]:

```
base_address_motor1 = 0x40000000
base_address_motor2 = 0x40001000
address_range = 0x1000
duty_cycle_addr_offset = 0x04
pulse_cycle_addr_offset = 0x08
duty_cycle_data_motor1 = 10200
pulse_cycle_data_motor1 = 5000
duty_cycle_data_motor2 = 10200
pulse_cycle_data_motor2 = 5000

# Motor 1
mmio_motor1 = MMIO(base_address_motor1, address_range)
#mmio_motor1.write(duty_cycle_addr_offset, duty_cycle_data_motor1)
#mmio_motor1.write(pulse_cycle_addr_offset, pulse_cycle_data_motor1)
#pulse_cycle_read_motor1 = mmio_motor1.read(pulse_cycle_addr_offset)

# Motor 2
mmio_motor2 = MMIO(base_address_motor2, address_range)
#mmio_motor2.write(duty_cycle_addr_offset, duty_cycle_data_motor2)
#mmio_motor2.write(pulse_cycle_addr_offset, pulse_cycle_data_motor2)
#pulse_cycle_read_motor2 = mmio_motor2.read(pulse_cycle_addr_offset)
```

In [12]:

```
base_address2=0x40001000


duty_cycle_data2 = 10000
pulse_cycle_data2 = 5000
#iic.mmio = MMIO(base_address2, address_range)
#iic.mmio.write(duty_cycle_addr_offset, duty_cycle_data2)
#iic.mmio.write(pulse_cycle_addr_offset, pulse_cycle_data2)
#iic.mmio.read(pulse_cycle_addr_offset)
```

In [ ]:

```
import matplotlib.pyplot as plt
Kp = 0.1
import time
frame_delay = 0.1  # Adjust this value as needed
while True:
    frame = vdma.readchannel.readframe()
    image = frame
    lane_check, lane_markings_result = detect_lane_markings(image)



#if lane_check == 1:
        # Calculate the deviation from the center of the lane
 #       lane_center = image.shape[1] // 2
 #       lane_mask = np.any(lane_markings_result, axis=2)
 #       lane_pixels = np.where(lane_mask)[1]
  #      if len(lane_pixels) > 0:
  #          lane_center_actual = np.mean(lane_pixels)
  #          deviation = lane_center - lane_center_actual

            # Adjust motor control based on deviation
            # Proportional control: adjust duty cycle proportionally to the deviation
  #          duty_cycle_motor1 = duty_cycle_data_motor1 + Kp * deviation
  #          duty_cycle_motor2 = duty_cycle_data_motor2 - Kp * deviation

            # Clip the duty cycles to stay within valid range
  #          duty_cycle_motor1 = max(0, min(65535, duty_cycle_motor1))
  #          duty_cycle_motor2 = max(0, min(65535, duty_cycle_motor2))

            # Set motor control signals
  #          mmio_motor1.write(duty_cycle_addr_offset, int(duty_cycle_motor1))
  #          mmio_motor2.write(duty_cycle_addr_offset, int(duty_cycle_motor2))
#else:
            # No lane pixels detected, stop the motors
 #           mmio_motor1.write(duty_cycle_addr_offset, 0)
  #          mmio_motor2.write(duty_cycle_addr_offset, 0)



 #Detect stop sign
#stop_sign_result = detect_stop_sign(image)

#if(stop_sign_result == 1):
```

```
 #   iic.mmio.write(duty_cycle_addr_offset, 0)
  #  iic.mmio.write(pulse_cycle_addr_offset, pulse_cycle_data)
   # iic.mmio.write(duty_cycle_addr_offset, 0)
    #iic.mmio.write(pulse_cycle_addr_offset, pulse_cycle_data2)
    if (lane_check == 1):
        mmio_motor1.write(duty_cycle_addr_offset, duty_cycle_data_motor1)
        mmio_motor1.write(pulse_cycle_addr_offset, pulse_cycle_data_motor1)
        mmio_motor2.write(duty_cycle_addr_offset, duty_cycle_data_motor2)
        mmio_motor2.write(pulse_cycle_addr_offset, pulse_cycle_data_motor2)
        pass
    else:
        mmio_motor1.write(duty_cycle_addr_offset, 0)
        mmio_motor1.write(pulse_cycle_addr_offset, 0)
        mmio_motor2.write(duty_cycle_addr_offset, 0)
        mmio_motor2.write(pulse_cycle_addr_offset, 0)

    pass
time.sleep(frame_delay)
#overlay_image = cv2.addWeighted(image, 1, lane_markings_result, 0.5, 0)
#overlay_image = cv2.addWeighted(overlay_image, 1, stop_sign_result, 0.5, 0)

# Display results
#img2 = Image.fromarray(overlay_image, 'RGB')
#display(img2)
```

In [ ]:

```
import matplotlib.pyplot as plt
Kp = 0.1

frame = vdma.readchannel.readframe()
image = frame
lane_check, lane_markings_result = detect_lane_markings(image)

#if lane_check == 1:
        # Calculate the deviation from the center of the lane
 #       lane_center = image.shape[1] // 2
 #       lane_mask = np.any(lane_markings_result, axis=2)
 #       lane_pixels = np.where(lane_mask)[1]
  #      if len(lane_pixels) > 0:
  #          lane_center_actual = np.mean(lane_pixels)
  #          deviation = lane_center - lane_center_actual

            # Adjust motor control based on deviation
            # Proportional control: adjust duty cycle proportionally to the deviation
  #          duty_cycle_motor1 = duty_cycle_data_motor1 + Kp * deviation
  #          duty_cycle_motor2 = duty_cycle_data_motor2 - Kp * deviation

            # Clip the duty cycles to stay within valid range
  #          duty_cycle_motor1 = max(0, min(65535, duty_cycle_motor1))
  #          duty_cycle_motor2 = max(0, min(65535, duty_cycle_motor2))

            # Set motor control signals
  #          mmio_motor1.write(duty_cycle_addr_offset, int(duty_cycle_motor1))
  #          mmio_motor2.write(duty_cycle_addr_offset, int(duty_cycle_motor2))
#else:
            # No lane pixels detected, stop the motors
 #           mmio_motor1.write(duty_cycle_addr_offset, 0)
  #          mmio_motor2.write(duty_cycle_addr_offset, 0)



 #Detect stop sign
#stop_sign_result = detect_stop_sign(image)

#if(stop_sign_result == 1):
 #   iic.mmio.write(duty_cycle_addr_offset, 0)
  #  iic.mmio.write(pulse_cycle_addr_offset, pulse_cycle_data)
   # iic.mmio.write(duty_cycle_addr_offset, 0)
    #iic.mmio.write(pulse_cycle_addr_offset, pulse_cycle_data2)
```

```
if (lane_check == 1):
        mmio_motor1.write(duty_cycle_addr_offset, duty_cycle_data_motor1)
        mmio_motor1.write(pulse_cycle_addr_offset, pulse_cycle_data_motor1)
        mmio_motor2.write(duty_cycle_addr_offset, duty_cycle_data_motor2)
        mmio_motor2.write(pulse_cycle_addr_offset, pulse_cycle_data_motor2)
        pass
else:
        mmio_motor1.write(duty_cycle_addr_offset, 0)
        mmio_motor1.write(pulse_cycle_addr_offset, 0)
        mmio_motor2.write(duty_cycle_addr_offset, 0)
        mmio_motor2.write(pulse_cycle_addr_offset, 0)

overlay_image = cv2.addWeighted(image, 1, lane_markings_result, 0.5, 0)
#overlay_image = cv2.addWeighted(overlay_image, 1, stop_sign_result, 0.5, 0)

# Display results
img2 = Image.fromarray(overlay_image, 'RGB')
display(img2)
```

In [ ]:

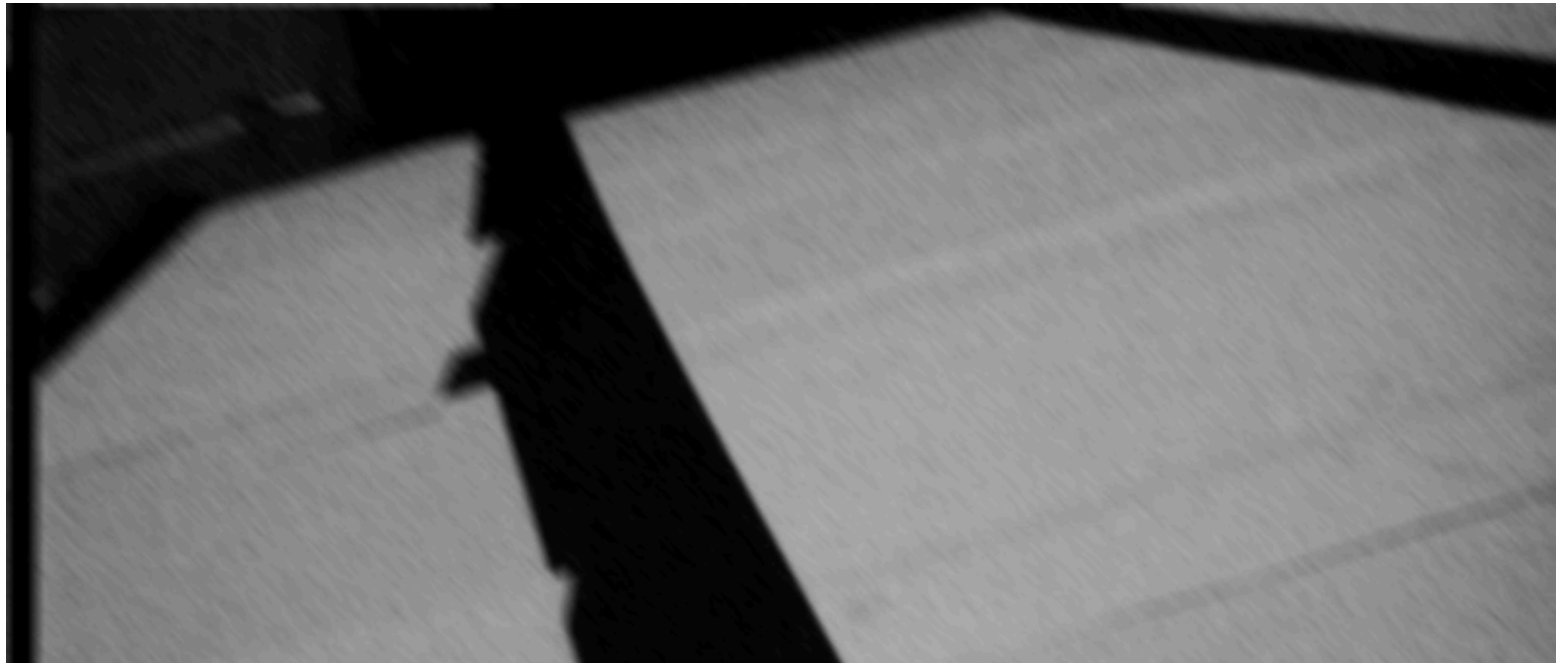
```
import matplotlib.pyplot as plt
import time

frame_delay = 0.1  # Adjust this value as needed
max_duty_cycle = 10200
max_pulse_cycle = 5000

while True:
    frame = vdma.readchannel.readframe()
    image = frame
    lane_check, lane_markings_result = detect_lane_markings(image)

    if lane_check == 1:
        # Lane detected, move forward
        mmio_motor1.write(duty_cycle_addr_offset, duty_cycle_data_motor1)
        mmio_motor1.write(pulse_cycle_addr_offset, pulse_cycle_data_motor1)
        mmio_motor2.write(duty_cycle_addr_offset, duty_cycle_data_motor2)
        mmio_motor2.write(pulse_cycle_addr_offset, pulse_cycle_data_motor2)
    else:
        # No lane detected, stop and turn
        mmio_motor1.write(duty_cycle_addr_offset, 0)
        mmio_motor1.write(pulse_cycle_addr_offset, 0)
        mmio_motor2.write(duty_cycle_addr_offset, max_duty_cycle)  # Max duty cycle for turning left
        mmio_motor2.write(pulse_cycle_addr_offset, max_pulse_cycle)

    time.sleep(frame_delay)
```

In [*]:

```
import matplotlib.pyplot as plt
import time

frame_delay = 0.1  # Adjust this value as needed
max_duty_cycle = 10100
max_pulse_cycle = 5000
count = 0
# Define the maximum duration for turning left or right (in seconds)
max_turn_duration = 0.3  # Adjust this value as needed

# Initialize variables for tracking turn duration and current state
turn_start_time = None
current_state = "forward"  # Initial state

while True:
    frame = vdma.readchannel.readframe()
    image = frame
    lane_check, lane_markings_result = detect_lane_markings(image)

    if lane_check == 1:
        # Lane detected, move forward
        mmio_motor1.write(duty_cycle_addr_offset, duty_cycle_data_motor1)
        mmio_motor1.write(pulse_cycle_addr_offset, pulse_cycle_data_motor1)
        mmio_motor2.write(duty_cycle_addr_offset, duty_cycle_data_motor2)
        mmio_motor2.write(pulse_cycle_addr_offset, pulse_cycle_data_motor2)

        # Reset the turn start time and state if lane is detected
        turn_start_time = None
        current_state = "forward"
    else:
        # No lane detected, stop and turn

        # Check if the turn has already started
        if turn_start_time is None:
            # Start the timer for turn duration and set the appropriate state
            turn_start_time = time.time()
            if current_state == "forward" or current_state == "right":
                current_state = "left"
            elif current_state == "left":
                current_state = "right"

        # Check if the turn duration has exceeded the maximum
        if time.time() - turn_start_time < max_turn_duration:
```

```
            # Continue turning based on the current state
            if current_state == "left":
                mmio_motor1.write(duty_cycle_addr_offset, 0)
                mmio_motor1.write(pulse_cycle_addr_offset, 0)
                mmio_motor2.write(duty_cycle_addr_offset, max_duty_cycle)  # Max duty cycle for turning l
                mmio_motor2.write(pulse_cycle_addr_offset, max_pulse_cycle)
              #  count+=1
            elif current_state == "right":
                mmio_motor1.write(duty_cycle_addr_offset, max_duty_cycle)  # Max duty cycle for turning r
                mmio_motor1.write(pulse_cycle_addr_offset, max_pulse_cycle)
                mmio_motor2.write(duty_cycle_addr_offset, 0)
                mmio_motor2.write(pulse_cycle_addr_offset, 0)
              #  count+=1
        else:
            # Stop turning and reset the turn start time for the next turn
            mmio_motor1.write(duty_cycle_addr_offset, 0)
            mmio_motor1.write(pulse_cycle_addr_offset, 0)
            mmio_motor2.write(duty_cycle_addr_offset, 0)
            mmio_motor2.write(pulse_cycle_addr_offset, 0)
            turn_start_time = None
            count+=1
            #if count >= 10:
             #   break
    time.sleep(frame_delay)
```

In [15]:
```
# Display results
frame = vdma.readchannel.readframe()
image = frame


img2 = Image.fromarray(image, 'RGB')
display(img2)
```



In [ ]:

In [ ]: