

Nume: Tănase Silviu Ionuț

Specializarea: Automatică și Informatică Aplicată

Grupa: AIA 2.3B

Data: 17.04.2022

Proiect Electronică Digitală

1. Introducere.

Pentru acest proiect am ales ca temă crearea unei aplicații în limbajul de programare Python care, folosind Raspberry Pi 4 Model B 8 GB RAM cu o cameră conectată la acesta, va detecta o persoană atunci când aceasta va apărea în câmpul vizual al camerei.

Cuprins

1. Introducere.....	1
2. Despre Raspberry Pi.....	3
2.1. Descrierea familiei de microprocesoare/microcontrolere utilizate de Raspberry Pi.	3
2.2. Descrierea sistemului de dezvoltare utilizat.	3
2.3. Descrierea modului.....	5
2.4. Mijloace de dezvoltare și testare a aplicației.....	5
2.5. Imagini cu realizarea practică a aplicației.	7
2.6. Descrierea aplicației software.....	8
3. Bibliografie:	12

2. Despre Raspberry Pi.

2.1. Descrierea familiei de microprocesoare/microcontrolere utilizate de Raspberry Pi.

Familia Raspberry Pi prezintă un sistem pe un cip Broadcom (Broadcom SoC) cu un microprocesor compatibil cu ARM și o placă video integrată.

Viteza procesorului variază de la 700 MHz până la 1.4GHz pentru Pi 3 Model B+ sau 1.5GHz pentru Pi 4.

Memoria RAM variază de la 256 MB până la 8GB, Raspberry Pi 4 fiind singurul cu mai multă memorie de 1GB.

Cardurile SD în formă MicroSD sunt folosite pentru a stoca sistemul de operare și memoria programelor, totuși există unele modele care vin cu stocare eMMC și Raspberry Pi 4 poate să folosească un SSD (sau un HDD) conectat prin portul USB.

2.2. Descrierea sistemului de dezvoltare utilizat.

Raspberry Pi 4 Model B are un procesor quad core 64-bit ARM-Cortex A72 1.5GHz. Memoria RAM a acestuia poate fi de 1GB(în prezent nu se mai fabrică)/2GB/4GB/8GB. Pentru stocare poate folosi memorie în format MicroSD, SSD, HDD, stick USB.

Conectivitate: 2.4 GHz și 5.0 GHz IEEE 802.11b/g/n/ac wireless; Bluetooth 5.0 cu BLE; Gigabit Ethernet; 2 porturi USB 3.0 ; 2 porturi USB 2.0.

Video și sunet: 2 × micro HDMI (până la 4Kp60); 2-lane MIPI DSI display port; 2-lane MIPI CSI port cameră; 4-pole stereo audio și composite video port.

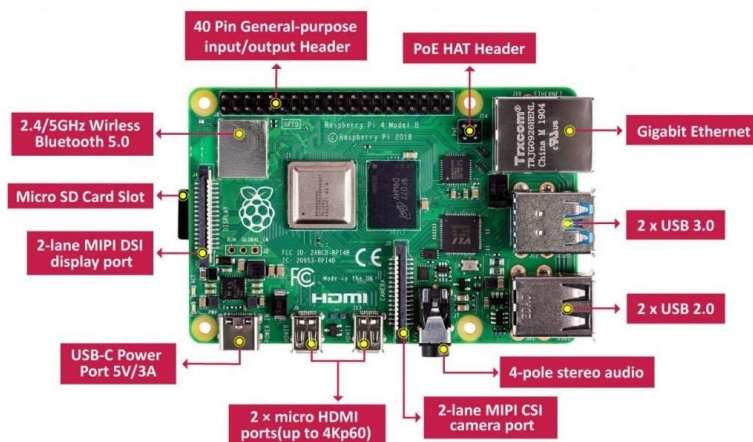


Figura 1

Pi4B are 28 BCM2711 GPIO-uri disponibile printr-un header standard Raspberry Pi de 40 de pini. Acest header este compatibil cu versiunile anterioare ale plăcilor Raspberry Pi care au 40-way header.

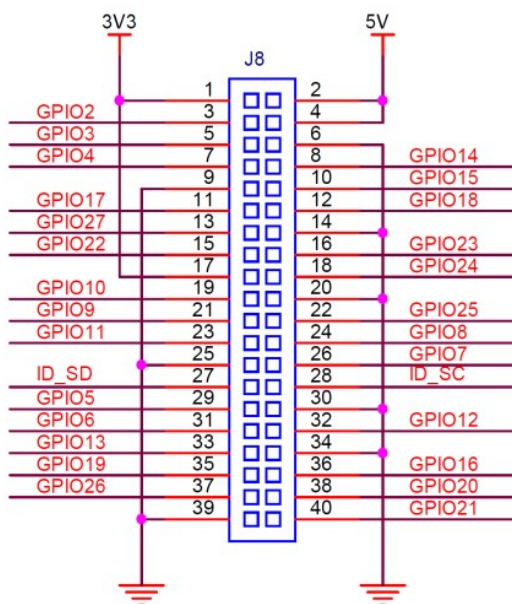


Figura 2

Pinii ID_SD și ID_SC: acești pini sunt rezervați pentru HAT ID EEPROM. La timpul boot-ului această interfață I2C va fi interogată pentru a se uita pentru EEPROM care identifică placa atașată și permite un setup automatic al GPIO-urilor (și opțional, drivere pentru Linux).

2.3. Descrierea modului.

Modulul folosit este o cameră pentru Raspberry Pi. Aceasta este compatibilă cu toate versiunile de Raspberry Pi, conectându-se prin interfața CSI a plăcuței de dezvoltare.

Caracteristici tehnice:

- 500 milioane pixeli;
- Chip OV5647;
- Senzor CCD de 1/4 inch;
- focalizare reglabilă;
- Diafragmă: f2.35;
- Distanță focală ajustabilă;
- Unghi de vizualizare de 160 grade;
- Rezoluție de până la 1080p;
- Tensiune de funcționare: 3.3V;
- Consum mic de curent.

2.4. Mijloace de dezvoltare și testare a aplicației.

Mediul de dezvoltare folosit este IDE-ul Thonny, acesta fiind inclus la instalarea sistemului de operare Raspbian. Thonny vine cu Python 3.7 încorporat.

Thonny are funcții menite să ajute programatorii începători să înțeleagă conceptele de programare. Cele mai proeminente dintre acestea sunt capacitatea de a parcurge codul, evaluarea pas cu pas a expresiei, vizualizarea stivei de apeluri și modurile de explicare a conceptelor de referințe și heap. Următoarele caracteristici pot fi văzute în utilizare, referindu-vă la Figura 3.

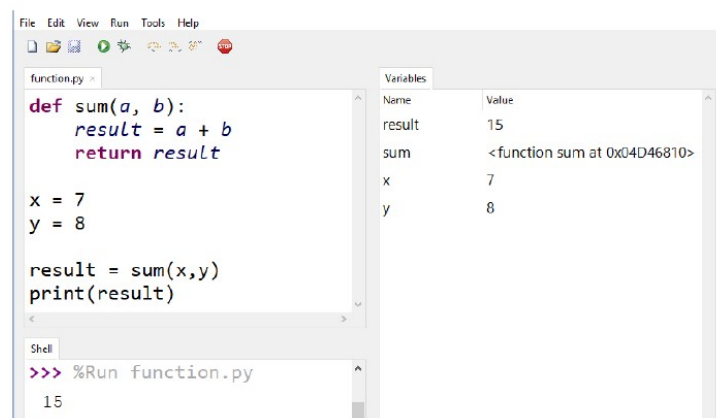


Figura 3

Funcționalitatea principală a oricărui IDE este editorul. Editorul lui Thonny oferă funcții de bază IDE, cum ar fi colorarea sintaxelor, potrivirea parantezelor, completarea codului, indentarea automată, indentarea blocurilor și comentarea blocurilor. De asemenea, oferă posibilitatea de a deschide mai multe fișiere sub diferite file. Evidențierea erorilor de sintaxă pentru paranteze deschise și ghilimele este, de asemenea, disponibilă.

Thonny oferă, de asemenea, un shell, care este o versiune îmbunătățită a shell-ului IDLE al lui Python. Spre deosebire de IDLE al lui Python, shell-ul lui Thonny este integrat în fereastra principală, deoarece acest lucru face ca utilizarea shell-ului să fie mai confortabilă pentru programatorul începător. Shell-ul folosește, de asemenea, diferite formataări pentru intrarea/ieșirea programului și operațiunile shell pentru a separa mai clar diferitele evenimente shell. Acest lucru poate fi văzut în Figura 4 – ieșirea expresiei este colorată în negru; intrarea expresiei este albastră, iar rezultatul evaluării expresiei este în aldine și albastru închis.

```
>>> input("What is 2 + 2? ")
    What is 2 + 2? 4
    '4'
>>>
```

Figura 4

Cea mai proeminentă caracteristică a lui Thonny este depanatorul său. Oferă posibilitatea de a executa programul utilizatorului pas cu pas, ajutându-l să înțeleagă elementele de bază ale modului în care este rulat codul scris în mașina virtuală a lui Python. Depanatorul funcționează prin întreruperea execuției codului la atingerea unei instrucțiuni. Depanarea se începe prin apăsarea butonului „Depanare script curent” sau cu combinația de taste „Control” + „F5”, care evidențiază automat prima afirmație. Utilizatorul poate alege apoi să pășească sau să treacă peste declarație. Executarea oricărei comenzi actualizează automat tabelul de variabile și ieșirea shell-ului în funcție de programul rulat și instrucțiunile evaluate.

Pe lângă faptul că este un IDE Python 3, Thonny este el însuși scris în Python 3. Thonny folosește două procese în timpul execuției – primul dintre ele este pentru interfața grafică cu utilizatorul (GUI), care se bazează pe framework-ul Python TkInter [17], al doilea este pentru back-end care gestionează execuția codului utilizatorului. Înainte de dezvoltarea oricărei caracteristici pentru Thonny, structura back-end-ului trebuie înțeleasă. Următoarea secțiune (exactă începând cu versiunea Thonny 2.1.16) oferă o privire de ansamblu asupra arhitecturii back-end-ului Thonny.

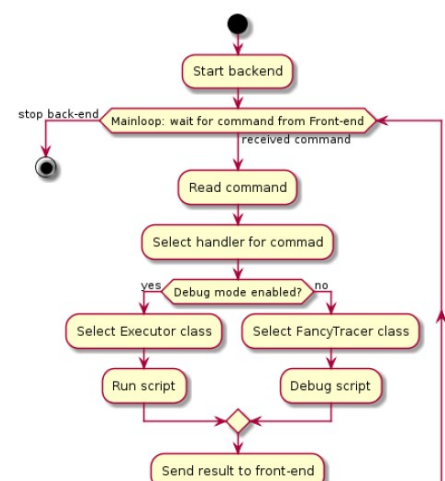
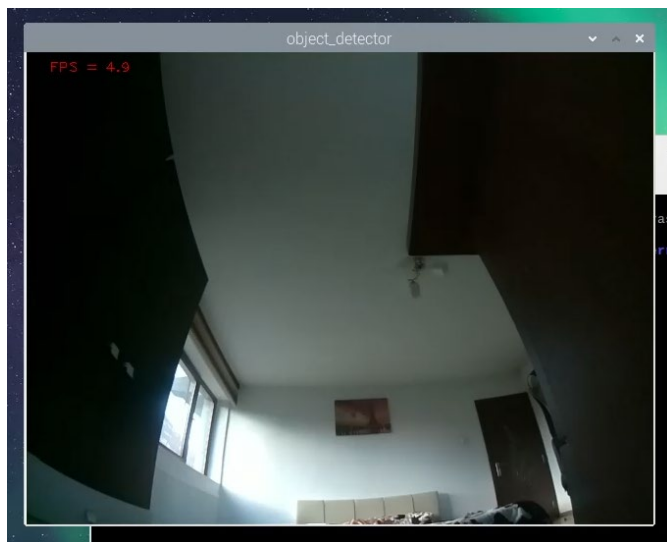
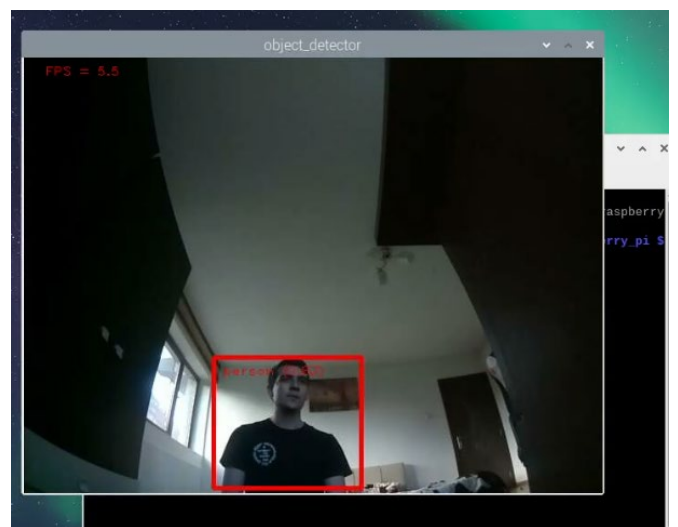


Figura 5

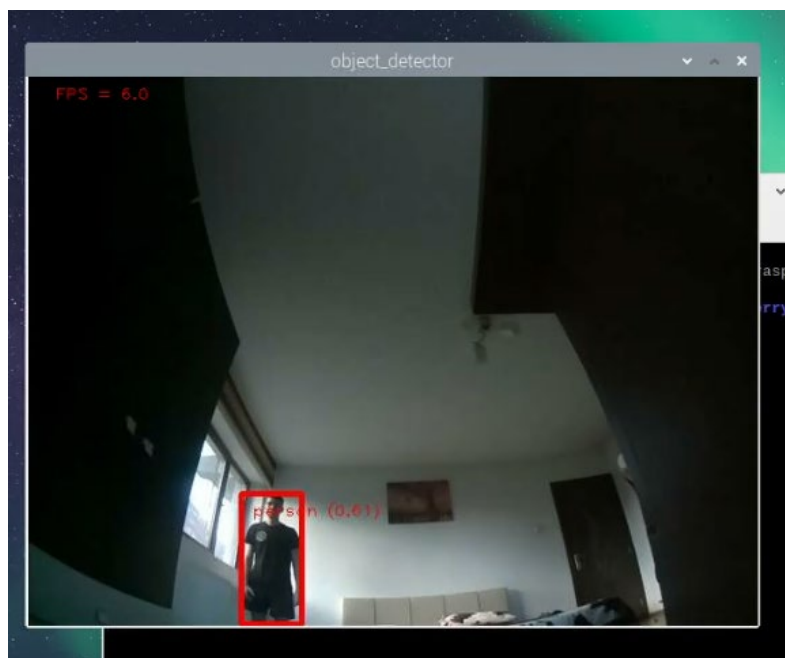
2.5. Imagini cu realizarea practică a aplicației.



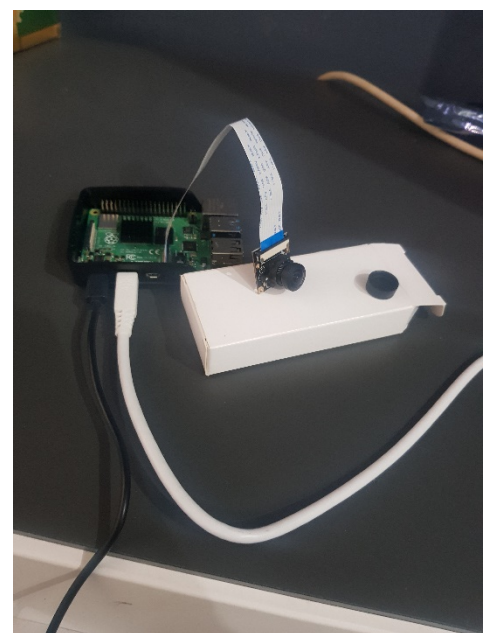
Captura 1



Captura 2



Captura 3



Captura 4

2.6. Descrierea aplicației software.

Pseudocod “detect.py”:

```
"""Scriptul principal pentru a rula detectarea."""
IMPORT argparse
IMPORT sys
IMPORT time

IMPORT cv2
from tfLite_support.task IMPORT core
from tfLite_support.task IMPORT processor
from tfLite_support.task IMPORT vision
IMPORT utils

DEFINE FUNCTION run(model: str, camera_id: int, width: int, height: int, num_threads: int,
    enable_edgetpu: bool) -> None:
    """Executa IN continuu interferente asupra imaginilor obtinute de la camera

    Args:
        model: Numele modelului de detectare a obiectului TFLite.
        camera_id: Id-ul camerei care este transmis catre OpenCV.
        width: Latimea cadrului capturat de catre camera.
        height: Inaltimea cadrului capturat de catre camera.
        num_threads: numarul de threads ale procesorului pentru a rula modelul.
        enable_edgetpu: Adevarat/Fals daca modelul este un model EdgeTPU."""

    # Variabile pentru calcularea cadrelor pe secunda (FPS).
    SET counter, fps TO 0, 0
    SET start_time TO time.time()

    # Incepere capturare video de la camera
    SET cap TO cv2.VideoCapture(camera_id)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)

    # Parametrii de vizualizare
    SET row_size TO 20 # pixeli
    SET left_margin TO 24 # pixeli
    SET text_color TO (0, 0, 255) # rosu
    SET font_size TO 1
    SET font_thickness TO 1
    SET fps_avg_frame_count TO 10

    # Initializarea modelului de detectare a obiectelor
    SET base_options TO core.BaseOptions(
        file_name=model, use_coral=enable_edgetpu, num_threads=num_threads)
    SET detection_options TO processor.DetectionOptions(
        max_results=3, score_threshold=0.3)
    SET options TO vision.ObjectDetectorOptions(
        base_options=base_options, detection_options=detection_options)
    SET detector TO vision.ObjectDetector.create_from_options(options)

    # Capturarea continua a imaginilor de la camera si rularea interferentei
    WHILE cap.isOpened():
        SET success, image TO cap.read()
        IF not success:
            sys.exit(
                'ERROR: Unable to read from webcam. Please verify your webcam settings.'
            )

        counter += 1
        SET image TO cv2.flip(image, 1)

        # Convertirea imaginii din BGR IN RGB asa cum este cerut de modelul TFLite
        SET rgb_image TO cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Crearea unui obiect TensorImage din imaginea RGB
        SET INPUT_tensor TO vision.TensorImage.create_from_array(rgb_image)

        # Rularea estimativa a detectarii obiectelor folosind modelul
        SET detection_result TO detector.detect(INPUT_tensor)

        # Desenarea punctelor cheie si marginilor pe imaginea de intrare
        SET image TO utils.visualize(image, detection_result)

        # Calcularea FPS-urilor
        IF counter % fps_avg_frame_count EQUALS 0:
            SET end_time TO time.time()
            SET fps TO fps_avg_frame_count / (end_time - start_time)
            SET start_time TO time.time()

        # Aratam FPS
        SET fps_text TO 'FPS TO {:.1f}'.format(fps)
        SET text_location TO (left_margin, row_size)
        cv2.putText(image, fps_text, text_location, cv2.FONT_HERSHEY_PLAIN,
            font_size, text_color, font_thickness)

        # Programul se opreste cand apasam tasta ESC.
        IF cv2.waitKey(1) EQUALS 27:
            break
        cv2.imshow('object_detector', image)

    cap.release()
    cv2.destroyAllWindows()

DEFINE FUNCTION main():
    SET parser TO argparse.ArgumentParser(
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument(
        '--model',
        help='Path of the object detection model.',
        required=False,
        default='efficientdet_lite0.tflite')
    parser.add_argument(
        '--cameraId', help='Id of camera.', required=False, type=int, default=0)
    parser.add_argument(
        '--frameWidth',
        help='Width of frame to capture from camera.',
        required=False,
        type=int,
        default=640)
    parser.add_argument(
        '--frameHeight',
        help='Height of frame to capture from camera.',
        required=False,
        type=int,
        default=480)
    parser.add_argument(
        '--numThreads',
        help='Number of CPU threads to run the model.',
        required=False,
        type=int,
        default=4)
    parser.add_argument(
        '--enableEdgeTPU',
        help='Whether to run the model on EdgeTPU.',
        action='store_true',
        required=False,
        default=False)
    SET args TO parser.parse_args()

    run(args.model, int(args.cameraId), args.frameWidth, args.frameHeight,
        int(args.numThreads), bool(args.enableEdgeTPU))
```

Captura 5

```
DEFINE FUNCTION main():
    SET parser TO argparse.ArgumentParser(
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument(
        '--model',
        help='Path of the object detection model.',
        required=False,
        default='efficientdet_lite0.tflite')
    parser.add_argument(
        '--cameraId', help='Id of camera.', required=False, type=int, default=0)
    parser.add_argument(
        '--frameWidth',
        help='Width of frame to capture from camera.',
        required=False,
        type=int,
        default=640)
    parser.add_argument(
        '--frameHeight',
        help='Height of frame to capture from camera.',
        required=False,
        type=int,
        default=480)
    parser.add_argument(
        '--numThreads',
        help='Number of CPU threads to run the model.',
        required=False,
        type=int,
        default=4)
    parser.add_argument(
        '--enableEdgeTPU',
        help='Whether to run the model on EdgeTPU.',
        action='store_true',
        required=False,
        default=False)
    SET args TO parser.parse_args()

    run(args.model, int(args.cameraId), args.frameWidth, args.frameHeight,
        int(args.numThreads), bool(args.enableEdgeTPU))
```

Captura 6

```
DEFINE FUNCTION main():
    SET parser TO argparse.ArgumentParser(
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument(
        '--model',
```

Captura 5

Pseudocod “utils.py”:

```
"""Funcții pentru afisarea rezultatelor detectării."""

import cv2
import numpy as np
from tfliite_support.task import processor

SET _MARGIN TO 10 # pixels
SET _ROW_SIZE TO 10 # pixels
SET _FONT_SIZE TO 1
SET _FONT_THICKNESS TO 1
SET _TEXT_COLOR TO (0, 0, 255) # red

DEFINE FUNCTION visualize(
    image: np.ndarray,
    detection_result: processor.DetectionResult,
) -> np.ndarray:
    """
    Desenează chenare pe imaginea de la intrare și o returnează.
    Args:
        image: Imaginea RGB de intrare.
        detection_result: Lista tuturor entităților "Detection" care trebuie vizualizate.

    Returns: Imaginea cu chenare de delimitare.
    """

    FOR detection IN detection_result.detections:

        # Trasează bounding_box
        SET bbox TO detection.bounding_box
        SET start_point TO bbox.origin_x, bbox.origin_y
        SET end_point TO bbox.origin_x + bbox.width, bbox.origin_y + bbox.height
        cv2.rectangle(image, start_point, end_point, _TEXT_COLOR, 3)

        # Trasează label și score
        SET category TO detection.classes[0]
        SET class_name TO category.class_name
        SET probability TO round(category.score, 2)
        SET result_text TO class_name + ' (' + str(probability) + ')'
        SET text_location TO (_MARGIN + bbox.origin_x,
                               _MARGIN + _ROW_SIZE + bbox.origin_y)
        cv2.putText(image, result_text, text_location, cv2.FONT_HERSHEY_PLAIN,
                    _FONT_SIZE, _TEXT_COLOR, _FONT_THICKNESS)

    RETURN image
```

Captura 7

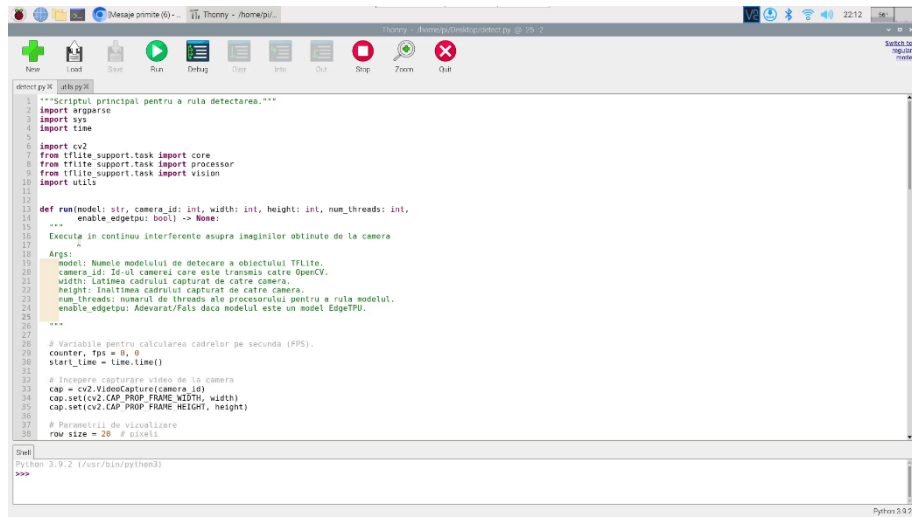
Librăriile utilizate sunt:

- argparse - folosită în funcția main, ajută la crearea unui program într-un mediu de linie de comandă într-un mod care pare nu numai ușor de codificat, ci și îmbunătățește interacțiunea.
- time – folosită la variabilele pentru calcularea cadrelor pe secundă
- opencv – folosită la capturarea imaginilor, crearea chenarelor specifice când avem o detectare.
- tensorflow – folosită la detecția propriu-zisă.
- numpy – folosită la operațiile matematice

Funcția "run" începe cu configurarea OpenCV-ului pentru a captura imagini de la cameră. Apoi inițializăm modelul pentru detecție Tensorflow și specificăm niște configurații opționale, precum score_threshold. Continuăm capturarea imaginilor până apăsăm tasta Esc și transmitem cadrele instanței de detectare pe care am creat-o mai devreme.

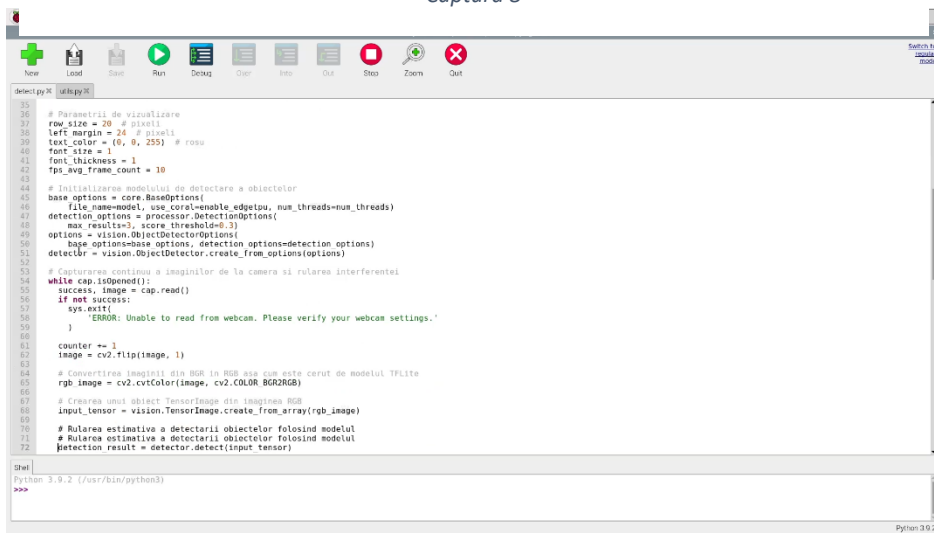
Pentru utilizarea aplicației am creat un mediu virtual, unde am instalat librăriile necesare (acest lucru este recomandat). Din acest mediu virtual trebuie doar să rulăm detect.py pentru a lansa aplicația.

Codul integral:



```
1 """Scriptul principal pentru a rula detectarea."""
2 import argparse
3 import sys
4 import time
5
6 import cv2
7 from tflite.support.task import core
8 from tflite.support.task import processor
9 from tflite.support.task import vision
10 import utils
11
12
13 def run(model: str, camera_id: int, width: int, height: int, num_threads: int,
14       enable_edgetpu: bool) -> None:
15     """
16     Executa in continuu inferente asupra imaginilor obtinute de la camera
17
18     Args:
19         model: Numele modelului de detectare a obiectului TFLite.
20         camera_id: Id-ul camerei care este transmis catre OpenCV.
21         width: Latimea cadrului capturat de catre camera.
22         height: Inaltimea cadrului capturat de catre camera.
23         num_threads: numarul de threads ale procesorului pentru a rula modelul.
24         enable_edgetpu: Adevarat/Fals daca modelul este un model EdgetPU.
25     """
26
27     # Variabile pentru calcularea cadrelor pe secunda (FPS).
28     counter = 0
29     start_time = time.time()
30
31     # Incepem capturarea video de la camera
32     cap = cv2.VideoCapture(camera_id)
33     cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
34     cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
35
36     # Parametri de vizualizare
37     row_size = 20 # pixeli
```

Captura 8



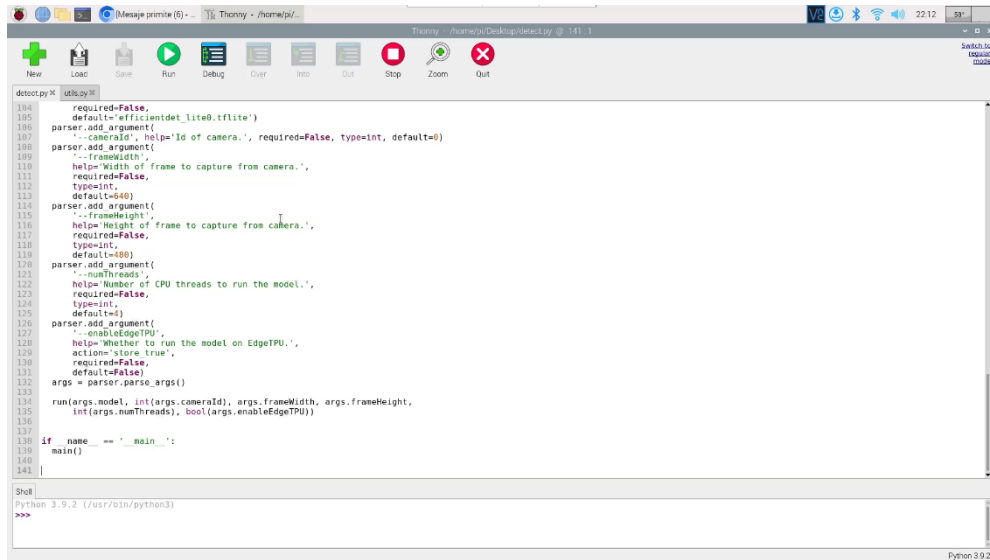
```
35 # Parametri de vizualizare
36 row_size = 20 # pixeli
37 left_margin = 24 # pixeli
38 text_color = (0, 0, 255) # rosu
39 font_size = 1
40 font_thickness = 1
41 fps_avg_frame_count = 10
42
43 # Initializarea modelului de detectare a obiectelor
44 base_options = core.BaseOptions()
45 file_name=model, use_coral=enable_edgetpu, num_threads=num_threads)
46 detection_options = processor.DetectionOptions(
47     max_results=1, score_threshold=0.3)
48 options = vision.ObjectDetectorOptions(
49     base_options=base_options, detection_options=detection_options)
50 detector = vision.ObjectDetector.create_from_options(options)
51
52 # Capturarea continua a imaginilor de la camera si rulara inferentelor
53 while cap.isOpened():
54     success, image = cap.read()
55     if not success:
56         sys.exit(
57             'ERROR: Unable to read from webcam. Please verify your webcam settings.'
58         )
59
60     counter += 1
61     image = cv2.flip(image, 1)
62
63     # Convertirea imaginii din BGR in RGB asa cum este cerut de modelul TFLite
64     rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
65
66     # Crearea unui obiect TensorImage din imaginea RGB
67     input_tensor = vision.TensorImage.create_from_array(rgb_image)
68
69     # Rulara estimativa a detectarii obiectelor folosind modelul
70     detection_result = detector.detect(input_tensor)
```

Captura 9

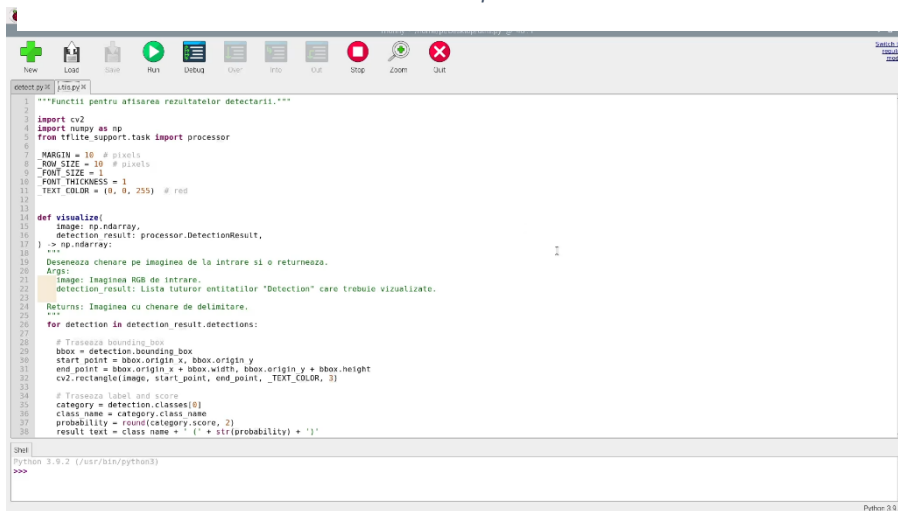


```
72 detection_result = detector.detect(input_tensor)
73
74 # Descrierea punctelor cheie si marginilor pe imaginea de intrare
75 image = utils.visualize(image, detection_result)
76
77 # Calcularea FPS-urilor
78 if counter > fps_avg_frame_count == 0:
79     end_time = time.time()
80     fps = fps_avg_frame_count / (end_time - start_time)
81     start_time = time.time()
82
83 # Arata FPS
84 fps_text = 'FPS: {}'.format(fps)
85 text_location = (left_margin, row_size)
86 cv2.putText(image, fps_text, text_location, cv2.FONT_HERSHEY_PLAIN,
87             font_size, text_color, font_thickness)
88
89 # Programul se opreste cand apasam tasta ESC.
90 if cv2.waitKey(1) &lt; 27:
91     break
92 cv2.imshow('object_detector', image)
93
94 cap.release()
95 cv2.destroyAllWindows()
96
97
98 def main():
99     parser = argparse.ArgumentParser(
100         formatter_class=argparse.ArgumentDefaultsHelpFormatter)
101     parser.add_argument(
102         '--model',
103         help='Path of the object detection model.',
104         required=True,
105         default='efficientdet_lite0.tflite')
106     parser.add_argument(
107         '--camera_id', help='Id of camera.', required=False, type=int, default=0)
108     parser.add_argument(
109         '--frames_count',
```

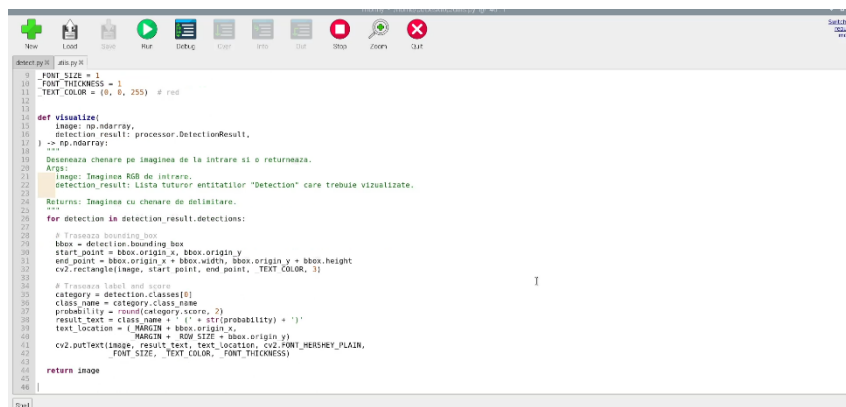
Captura 10



Captura 11



Captura 12



Captura 13

3. Bibliografie:

- <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf>
- <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf>
- <https://www.tensorflow.org/lite>
- <https://pypi.org/project/opencv-python/>
- https://comserv.cs.ut.ee/home/files/leemet_informaatika_2018.pdf?study=ATILoputoo&reference=57D36B3D851E10E6FC750A19E3479D643D8A0FB7