Moving Objects Detection

Daian Petre-Mihai

2023, semester I, third year of studies

Outline

1 Introduction

2 Context

This project was developed as a laboratory assignment for the "Structure of Computer Systems Lecture" in the Fall of 2023. I have taken this project as a challenge to learn more about the video processing field. Making contact with this new topic of study will create many new solutions that could be used in future projects.

Our world is increasingly interconnected through technology and the ability to monitor and analyze moving objects is of significant importance. Whether it's for security, surveillance, traffic control or even encountering objects, the capacity to identify and track moving objects is an essential aspect of many contemporary systems. This project called "Moving Objects Detection, leverages the power of web cameras and Python programming to address the use cases.

3 Motivation

The personal motivation lies in the ambition to learn, understand and work with video processing solutions that sustain the nowadays needs of the customers. The general motivations behind this projects stems from the ever-growing demand for accurate. real-time information about the position and behavior of moving objects.

The projects not only fulfills the need for reliable object tracking but also refines the technology. By utilizing a readily available and cost-effective tool such as a web camera and a programming language as versatile as Python, it makes object detection accessible to a wider audience.

Additionally, the project serves as an educational and exploratory platform. It allows me as student to delve into computer vision, imagine processing and machine learning in a hands-on, practical way.

4 Objectives

The primary goal of "Moving Objects Detection" project is to develop a system that can identify and track moving objects in a real-time using a web camera and Python tools, and to process this information for specific applications. Knowing this, the project's objectives can be stated as:

- To develop a robust object detection system that can identify and locate moving objects with the camera's field of view.
- Ensure that the system can process and analyze the object's position and movement in real-time, providing up to the moment data for applications.
- Implement the capability to track multiple moving objects simultaneously, allowing for the analysis of complex scenarios.
- Create a user-friendly interface that displays the detected objects and their position, making it easy for users to interpret the data.

5 Bibliographic Study

5.1 Object Tracking and Detection

At the heart of computer vision lies the capability to identify and follow objects within digital media. Object tracking and detection are two intertwined techniques that enable machines to see, understand, and interact with their environment, much like the human visual system.

Object detection is the process of recognizing and locating specific objects within an image or video frame. It's akin to a machine's ability to say, "There's a cat at this particular location in the image."

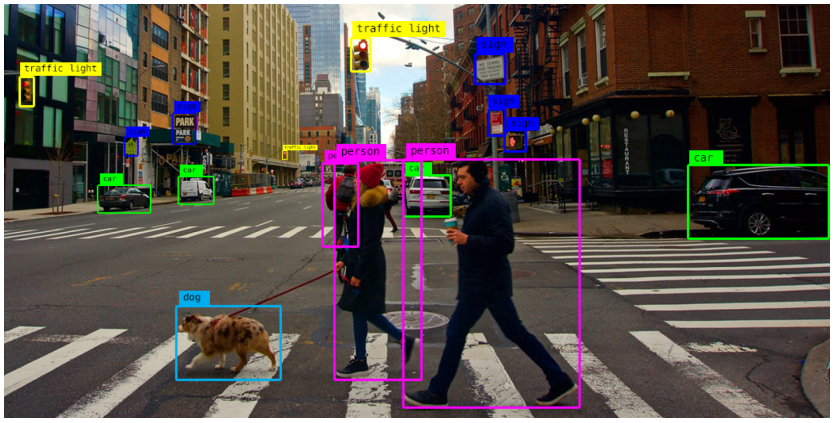


Figure 1: Object Detection Resulted Frame. Source: alwaysai.co

With the rise of neural networks, models like R-CNN, YOLO, and SSD have become the gold standard. They automatically learn features from data and have significantly improved detection accuracy. With the recently development of deep learning in the artificial intelligence domain, this tools employ the GPU so that it makes a lot more easier to be used and ease the work of the developer.

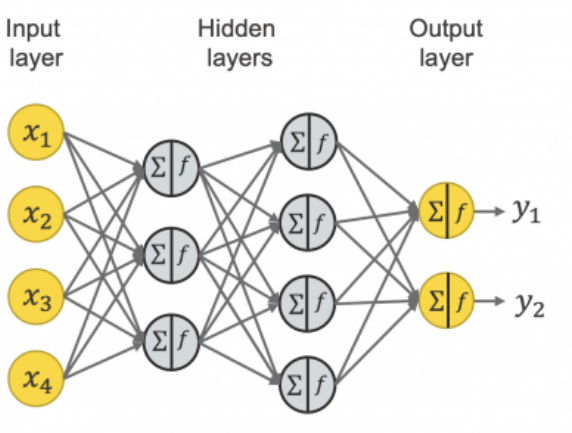


Figure 2: An example of Neural Network. Source: knime.com

After detecting an object, the next challenge is to keep track of its movement across frames. It's like a machine saying, "The cat that was here is now moving to the right." Instead of treating detection and tracking as separate processes, they can be integrated. Detection runs periodically, and tracking ensures smooth transitions between detections.

An object might get confused with another. Advanced trackers maintain unique IDs and use appearance models to prevent such switches.If an object is hidden for a while and then reappears, maintaining its identity is tough. Modern trackers use short-term memory mechanisms to handle such scenarios.

These two concepts have improved and expanded some well-known areas of technology like Robotics, where robots use object tracking and detection to interact with the surrounding's, Medical Imaging where the detection of anomalies in medical scans could aid in early diagnosis and treatment of diseases and Drones where are used to avoid obstacles in the environment and to follow or reach the target.

5.2 Traditional used methods

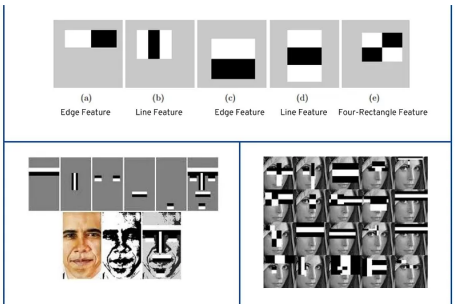
Before the advent of deep learning, several traditional methods laid the foundation for object tracking and detection. These methods are often rule-based and involve more handcrafting and feature engineering. The following three methods describe the object detection methods:

- *Haar Cascades*: A machine learning-based approach where the cascade function is trained from lots of positive and negative images. It was a pioneering technique for face detection.
- *Histogram of Oriented Gradients (HOG)*: This method involves dividing the image into small connected regions, called cells, and for each cell computing a histogram of gradient directions or edge orientations for the pixels within the cell.
- *Template Matching*: This approach searches for parts of an image that match a template image and is simple but can be effective in controlled environments.

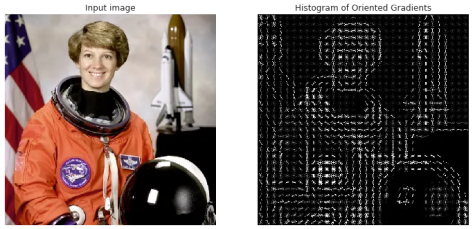
And these are three methods used in object tracking:

- *Mean Shift*: A simple and effective technique that involves iterating over candidate windows according to a similarity measure until convergence.
- *Kalman Filter*: This algorithm predicts the future location of an object based on its current state and past movements, assuming the object moves in a predictable manner.
- *Optical Flow*: It uses the apparent motion of objects between two consecutive frames due to the movement of the object or camera.

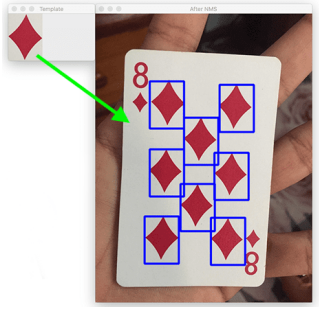
Figure 3: Traditional used methods



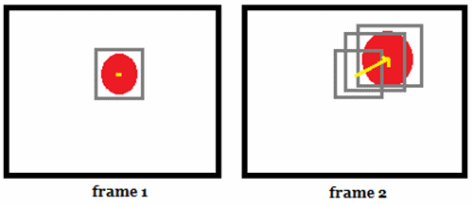
(a) Haar Cascade. Source: ai.plainenglish.io



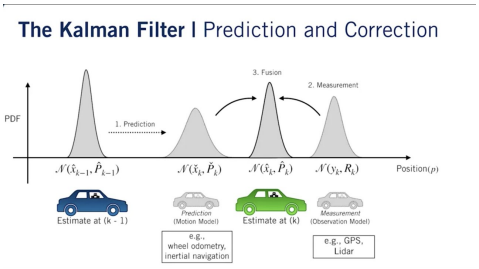
(b) Histogram of Oriented Gradients. Source: medium.com



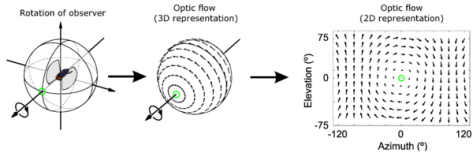
(c) Template matching. Source: pyimagesearch.com



(d) Mean Shift. Source: ni.com



(e) Kalman Filter. Source: yt/Machine Learning TV



(f) Kalman Filter. Source: wikipedia.com

5.3 Deep Learning-Based Methods - YOLO

5.3.1 Introduction

YOLO (You Only Look Once) is a state-of-the-art, real-time object detection system that has revolutionized the field of computer vision. Initially introduced by Joseph Redmon et al. in 2016, YOLO stands out for its speed and accuracy, making it an ideal choice for applications requiring real-time processing, such as video surveillance and autonomous vehicles.

5.3.2 How it works?

YOLO adopts a unique approach by dividing the input image into a grid. Each grid cell is responsible for predicting bounding boxes and confidence scores for objects within the cell. This single-pass prediction contrasts with traditional two-stage detectors, significantly speeding up the process. YOLO also applies non-maximum suppression to refine its predictions, reducing redundancy and improving accuracy.

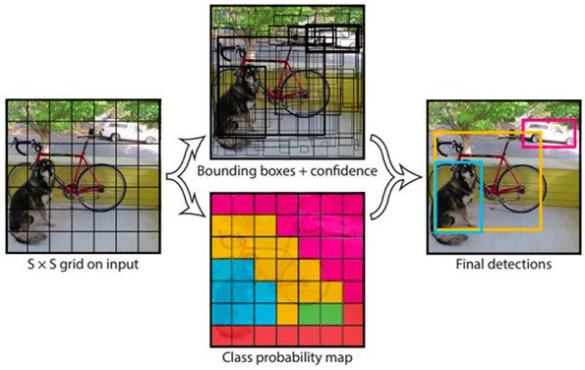


Figure 4: Yolo Design. Source: pyimagesearch.com

5.3.3 Evolution

Over the years, YOLO has seen several iterations, each improving upon its predecessor. From YOLO v2, which introduced anchor boxes and batch normalization, to YOLO v7, which brought in advancements like focal loss and higher resolution image processing, the evolution of YOLO has been marked by significant enhancements in speed and accuracy. Each version aims to balance the trade-off between detection speed and accuracy, making YOLO adaptable to various real-world applications.

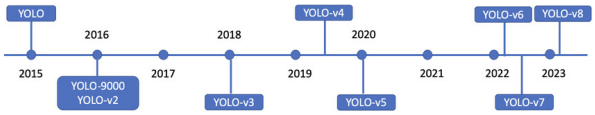


Figure 5: Yolo Evolution. Source: researchgate.net

5.3.4 Limitations and Considerations

While YOLO is a powerful tool for object detection, it does have limitations. It can struggle with detecting small objects and is sensitive to variations in object scale and environmental conditions. Additionally, the computational intensity of YOLO models, especially the more advanced versions, can be a consideration for deployment on resource-constrained devices.

5.4 Euclidean Distance Tracker

This subsection delves into the method of tracking objects across video frames using Euclidean distance. It builds upon the concept of object detection, particularly using YOLOv3, to track multiple objects by referencing their positions in previous frames. This approach assigns unique labels to each detected object, facilitating their tracking over time.

YOLOv3 is employed to detect objects in each frame, providing coordinates (x, y, width, height) for each object.The tracking is achieved by comparing the coordinates of objects in the current frame with those in the previous frame. This comparison is based on the Euclidean distance, a straightforward measure of the straight-line distance between two points in space.

The core of this tracking method lies in the calculation of Euclidean distance between objects across frames. If the distance between the same object in consecutive frames is less than a set threshold (e.g., 50 units), it is identified as the same object. This process involves using Numpy library from Python for distance calculation .

Effectiveness of this tracking method hinges on the consistent detection of objects in every frame. When objects are detected reliably, the tracking algorithm, based on Euclidean distance, performs efficiently in maintaining the continuity of object labels across frames.

Figure 6: Euclidean Distance Tracker. Source: medium.com

5.5 OpenCV

5.5.1 Introduction

OpenCV (Open Source Computer Vision Library) is a highly acclaimed, open-source computer vision and machine learning software library. Initially developed by Intel, it has grown into a global community-driven project. OpenCV is designed to provide a common infrastructure for computer vision applications, accelerating the use of machine perception in commercial products. Being open-source, it is free for both academic and commercial use under a BSD license.

Figure 7: OpenCV Logo. Source: wikipedia.com

5.5.2 Comprehensive Functionality

OpenCV specializes in real-time image processing and includes hundreds of computer vision algorithms. Its capabilities span across various domains including: OpenCV specializes in real-time image processing and includes hundreds of computer vision algorithms. Its capabilities span across various domains including:

- **Basic Image Processing**: Functions like filtering, transformations, and geometric image transformations.
- **Feature Detection and Description**: Techniques to identify and describe visual features such as edges, corners, and objects.
- **Object Detection**: Advanced algorithms for detecting specific objects like faces, eyes, or cars.
- **Video Analysis**: Includes motion estimation, background subtraction, and object tracking algorithms.
- **Camera Calibration and 3D Reconstruction**: Tools for 3D modeling, including stereo imaging and structure from motion.
- **Machine Learning**: Integrated with tools for classification, regression, and clustering.

5.5.3 Core Architecture

Modular Structure: OpenCV is organized into several modules, each focusing on different aspects of computer vision and image processing. These modules include core functionality, image processing, video analysis, camera calibration, 2D and 3D features framework, object detection, machine learning, and GUI operations.

Optimized Performance: Many of the underlying algorithms in OpenCV are optimized for performance, particularly for real-time applications. It uses advanced techniques like multi-threading and hardware acceleration (e.g., via CUDA and OpenCL) to enhance computational efficiency.

5.5.4 Image Processing

Handling Images: At its core, OpenCV reads images as multi-dimensional arrays (using NumPy in Python). Each pixel value can be manipulated or accessed for processing tasks like filtering, edge detection, and color space conversions.

Algorithms Implementation: OpenCV implements a variety of algorithms for tasks such as smoothing, thresholding, gradients, histograms, and transformations. These algorithms are designed to be both efficient and easy to use.

5.5.5 Computer Vision Techniques

Feature Detection and Matching: OpenCV provides algorithms for detecting and matching key points in images, which are essential for tasks like object recognition and motion tracking.

Object Detection: It includes pre-trained classifiers for face and eye detection, among others, and supports custom classifiers for detecting any type of objects.

Machine Learning Integration:OpenCV integrates with machine learning libraries, allowing for sophisticated image classification and recognition tasks.

5.5.6 Video Analysis

Capture and Processing: OpenCV can capture video from various sources, including live streams and video files. It processes video frames in real-time, enabling applications like motion detection, object tracking, and background subtraction.

Real-Time Interaction: The library allows for real-time interaction with video feeds, enabling applications like augmented reality or interactive installations.

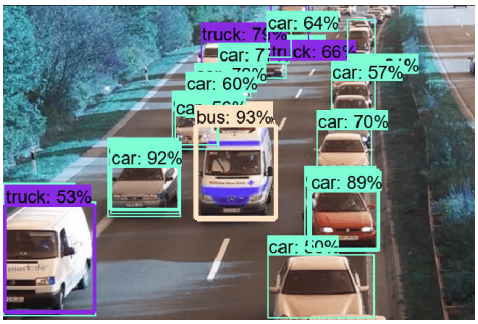


Figure 8: OpenCV Autovehicles detection. Source: researchgate.net

6 Design and Analysis

After the study of the previously mentioned topics, design and analysis section of the project will focus on explaining the structure, functionality and underlying the principles of the project.

6.0.1 System Design

First thing that has to be discussed is the **Architecture Overview** of the project. The architecture of the object tracking system is designed to seamlessly integrate various components, ensuring efficient and accurate object detection and tracking. At its core, the system comprises three primary modules: the camera interface, object detection, and tracking modules, each playing a crucial role in the overall functionality.

Camera Interface is the initial point of interaction with the physical environment. It captures real-time video data, which is essential for object detection and tracking. The camera interface is configured to optimize the quality and characteristics of the video feed, ensuring that the subsequent modules receive clear and usable input.

Once the video feed is captured, the **Object Detection** module comes into play. This module analyzes the video frames to identify and locate objects of interest within the frame. It employs

sophisticated algorithms to distinguish between different objects and their features, a critical step for accurate tracking.

After objects are detected, the **Tracking Modules** take over. These modules are responsible for continuously monitoring and following the movements of identified objects across the video frames. The system incorporates two different tracking approaches, offering flexibility and adaptability in various scenarios.

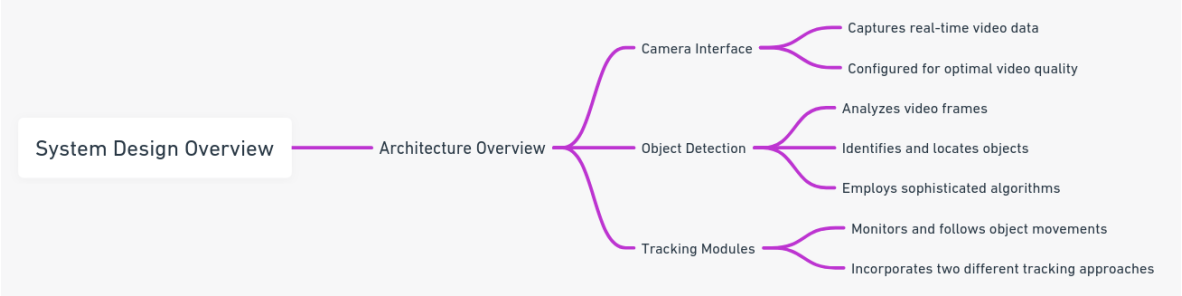


Figure 9: System design

6.0.2 Modules Description

The entire work has to be divided in three modules which will play the role of components along a capturing and processing line. Each of these take the input from another component and export the output to another component. The modularity non-functional feature of the project is ensured by this work flow.

Camera Configuration: The *window_conf* function plays a vital role in the initial setup of the camera, specifically tailored for object detection tasks. It meticulously adjusts the camera's resolution to ensure clarity and detail in the captured video, which is essential for accurate object detection. Additionally, it configures the frame rate to achieve a balance between smooth video playback and efficient processing. The brightness setting is also modified to adapt to different lighting conditions, ensuring consistent visibility of objects. This careful tuning of camera settings by *window_conf* is fundamental in ensuring that the camera feed is of the highest quality, a prerequisite for effective object detection and tracking.

Video Frame Generation: The *generate_video_frame*s function is central to the data acquisition process. It starts by clearing any existing data in the designated storage folder, ensuring that only the most recent frames are processed. The function then captures images continuously from the camera feed over a specified duration. These images are converted to grayscale, simplifying the data and reducing the computational load during processing. Each image is saved as an individual frame in the designated folder. This process is crucial as it provides the raw video frames necessary for the subsequent object detection and tracking phases, forming the backbone of the data acquisition process.

Object Tracking: The system employs two distinct functions for object tracking: *track_objects_simple* and *track_objects_yolo*. The *track_objects_simple* function uses a basic approach of background subtraction and contour detection, effective in environments with minimal background movement and noise. It is best suited for scenarios where object movements are predictable and the background is static. On the other hand, *track_objects_yolo* utilizes the advanced YOLO (You Only Look Once) algorithm, known for its speed and accuracy in real-time object detection. This method is capable of identifying and tracking multiple objects simultaneously, even in complex and dynamic environments. It offers superior performance in varied scenarios, making it ideal for applications that demand high accuracy and robustness in object tracking. These two tracking methods provide the system with flexibility and adaptability, catering to a wide range of object tracking scenarios.

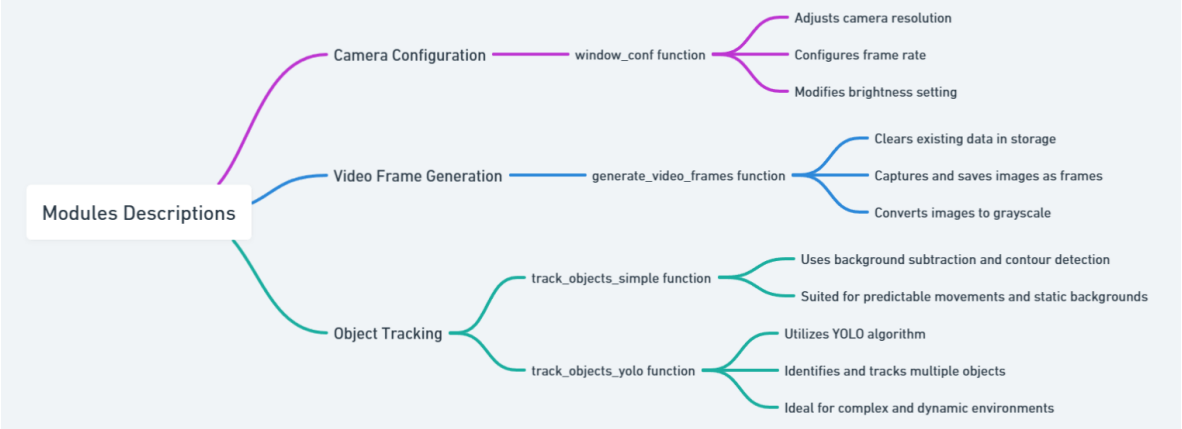


Figure 10: Modules description.

7 Implementation

After the previously discussion about the architecture and the design of the project, the next step is to choose the tools to be used. Knowing that the library OpenCV will play a major role in the goal achieving, Python language becomes the principal toll used. In this phase the programming environment it is also a decision to take so the IDE from JetBrains PyCharm makes the work a pleasure and a lot easier. The current version of Python enabled in this project is 3.11 that ensures all the posibilities.

Figure 11: Python 3.11 and PyCharm

7.1 main.py

The Python script starts with the main file of the project called *main.py* in which the principal function *track_object_yolo()* is called.

Listing 1: main.py

```
1    from video_manager import track_objects_yolo
2    track_objects_yolo()
```

The other files are *video_manager.py* and *tracker.py*. We will start first with the *video_manager.py* to describe the code developed inside. This file it is designed in several functions similar to the modules describe in the last section. Each function serves a specific purpose in the process.

There are some libraries that have to be used in order to complete the process.

7.2 video_manager.py

Listing 2: Imported libraries

```
1   import cv2
2   import time
3   import os
4   from tracker import *
5   from ultralytics import YOLO
6   import math
```

First import *cv2* it is the library of the *OpenCV* standard and will be mainly used during the video capturing process and record manipulation. *time* import will be used in the *generate_video_frames* function which will capture a short video for only 4 seconds. Next, *os* import it is used in files and folders management, *tracker* import it servers as the second file which will be explained later, *ultralytics* import from YOLO it is the specificic library that will helped during detection and tracking process. And finally, *math* it is useful when we have to deal with mathematical operations.

Listing 3: Cleaning the frames folder

```
1   def clean_folder(path):
2       for filename in os.listdir(path):
3           os.remove(os.path.join(path, filename))
```

It was said earlier that there is a function that will capture for 4 seconds all the frames from a video. All those frames are saved in a local directory everytime the function it is called and to have a better management of the folder, a function for cleaning the folder everytime the code is running it is used. The path of the folder it is specified in the calling function which knows the path.

Listing 4: Viewing window configuration

```
1   def window_conf():
2       width = 640
3       height = 480
4       cap = cv2.VideoCapture(0)
5       cap.set(3, width)
6       cap.set(4, height)
7       cap.set(10, 150)
8       return cap
```

The purpose of this function is to configure and return a video capture object. The webcam it is set up with a specified width = 640, height = 480 and brightness = 150. At line 4 the *cap* object represents and instance of VideoCapture with value 0 representing the default camera.

On Listing 5, the function ***generate_video_frames*** has as parameter the *folder* where the frames extracted from the record will be saved. First, it calls the *clean_folder* function and checks whether the passed folder exists and if not it creates a new one. After that, it gets the *cap* object from the window configuration function. It initializes an empty list called *frames* to save the obtained frames and also stores the current time in *start_time* to know when the 4 seconds elapsed. Starts the while loop for a period of 4 seconds given by the current time and start time.

Next, it reads a frame from the capture object into a tuple where *success* is a Boolean indicating where it was read successfully and *img* refers the captured frame. The algorithm needs the frames to be in grayscale so the frames are converted in the next instruction using the static method *cvtColor* from *cv2* library specifying also the targeted color. After this, the resulted frame it is appended it the list. After the loop ends, at line 17, the *cap* object it is released and all the elements from *frames* will be mapped to a file in the folder. At the end, all the opened windows are closed the the list it is retrieved from the function.

Listing 5: Generating frames from the record of 4 seconds

```
def generate_video_frames(folder):
clean_folder(folder)
if not os.path.exists(folder):
    os.makedirs(folder)

cap = window_conf()

frames = []
start_time = time.time()

# Capture time of 4 seconds
while time.time() - start_time < 4:
    success, img = cap.read()
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    frames.append(img_gray)

cap.release()

for i in range(len(frames)):
    cv2.imwrite(os.path.join(folder, f"frame_{i}.png"), frames[i])

cv2.destroyAllWindows()
return frames
```

On Listing 6, the function *track_objects_simple* it uses the traditional methods to identify and track objects in a record. First, we get the video capture object from the window configuration function. On the line 5, we intiliatize an object detector using the *MOG2 (Mixture of Gaussian)* background subtraction method. This technique it is used to separate moving objects (foreground) from a static or slowly changing background in video streams. It models each pixel as a mixture of multiple Gaussian distributions, continuously adapting to changes in the scene to identify and track foreground elements effectively.

The first parameter *history=100* it sets the number of last frames that affect the background model, and *varThreshold=40* sets the threshold on the squared Mahalanobis distance to decide whether it's background or foreground. Mahalanobis Distance is a statistical tool used to measure the distance between a point and a distribution. It is a powerful technique that considers the correlations between variables in a dataset, making it a valuable tool in various applications such as outlier detection, clustering, and classification.

Next, an Euclidean Distance Tracker Object it is created as *tracker*. This module will be explained better on the next subsection *tracker.py*. Going into the while loop which is checked infinitely until a break instruction occurs, we get the same tuple, the boolean *success* and the frame *img* from the capture object. We enter the object detection block of code. On the line 13 the MOG2 object detector it is applied to the frame to get the foreground *mask*. Then, it is applied a binary threshold to the *mask*. Pixels with values above 254 will be set to 255 (white), and all others to 0 (black), enhancing the distinction between foreground and background.

To know better where the object it is localized on the frame, the program finds contours in the binary mask. *cv2.RETR_TREE* retrieves all contours and creates a full family hierarchy, and *cv2.CHAIN_APPROX_SIMPLE* compresses horizontal, vertical, and diagonal segments, leaving only their end points. Am empty list *detections* it is initialized to store the detected objects. Now, for every countour found in the mask, the area occupied in the frame it is computed and if it is greater than 200 pixels, the bounidng box it is deconstructed using the *boundingRect* method from *cv2*, othewise the contour it is filtered.

In the object tracking block of code, the tracker store in *boxes_id* the identified boxes with their IDs. It iterates each object from the boxes list unpacking the position and dimensions of it. On the next lines, the id of the tracked object it is put on the frame and the rectangle around the object it is drawn. The line 32 shows the raw capture recorded from the webcam and the 33 line displays the mask obtained from it. If the *q* key it is pressed, then the capturing process ends and the capture object the and window are released.

Listing 6: Basic tracking of objects without machine learning

```
1     def track_objects_simple():
2     cap = window_conf()
3
4     # Object detection from stable camera
5     object_detector = cv2.createBackgroundSubtractorMOG2(history=100,
6                                                          varThreshold=40)
7     tracker = EuclideanDistTracker()
8
9     while True:
10        success, img = cap.read()
11
12        # Object detection
13        mask = object_detector.apply(img)
14        _, mask = cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY)
15        contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
16        detections = []
17        for cont in contours:
18            # Compute area and remove small elements
19            area = cv2.contourArea(cont)
20            if area > 200:
21                x, y, w, h = cv2.boundingRect(cont)
22                detections.append([x, y, w, h])
23
24        # Object tracking
25        boxes_ids = tracker.update(detections)
26        for box_id in boxes_ids:
27            x, y, w, h, id = box_id
28            cv2.putText(img, str(id), (x, y - 15), cv2.FONT_HERSHEY_PLAIN, 2,
29                                                          (255, 0, 0), 2)
30            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 3)
31
32        cv2.imshow("Live Cam", img)
33        cv2.imshow("Mask", mask)
34        key = cv2.waitKey(30)
35        if key == ord('q'):
36            break
37    cap.release()
38    cv2.destroyAllWindows()
```

This time , in Listing 7., YOLO it is used to track and identify objects. As before, the video capture object it is initialized as *cap* where 0 argument means that the default camera it is used as video source. The dimensions of the camera are set next with *width* of 1600 pixels and *height* of 900 pixels.

Afterwards, the YOLO model it is initialized in instance *model* with the specified weights from the file *yolo8n.py* This file contains the pretrained weights for the YOLO model. A list of strings representing different classes that the YOLO model can detect(e.g., "person", "dog", "laptop", etc.) it is saved in the *classNames* instance.

Object detection and tracking starts with an infinite while loop created by the True boolean in order to continously capture frames from the webcam and to process them. A frame it is read from the webcam and therefore *success* is a boolean indicating if the frame was successfully read and *img* is the frame itself. The captured frame it is passed to the YOLO model for object detection in line 26 and the results are saved in *results*. The argument *stream=True* represents the scenario of handling video streams.

The coming for loop processes the detection results. The code iterates over the results *for r in results:* and further iterates over the detected boxes in each result *for box in boxes:*. It extracts and processes the bounding box coordinates *x1, y1, x2, y2* for each detected object. The ode then displays the top-right corner coordinates of the bounding box on the image. A rectangle it is drawn or a *bounding box* around the detected object. It also computes the confidence score of the detection. The class of the detected object it is determined and printed therefore.

Line 62 displays the processed frame with the detected object and their bounding boxes. The loop can be exited by pressing *q* key after which the webcam is released and all the windows closed.

Listing 7: Yolo tracking of objects with machine learning

```
    def track_objects_yolo():
    cap = cv2.VideoCapture(0)
    cap.set(3, 1600)
    cap.set(4, 900)

    model = YOLO("yolo-Weights/yolov8n.pt")

    classNames = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus",
                  "train", "truck", "boat", "traffic light", "fire hydrant",
                  "stop sign", "parking meter", "bench", "bird", "cat", "dog",
                  "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe",
                  "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee",
                  "skis", "snowboard", "sports ball", "kite", "baseball bat",
                  "baseball glove", "skateboard", "surfboard", "tennis racket",
                  "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl",
                  "banana", "apple", "sandwich", "orange", "broccoli", "carrot",
                  "hot dog", "pizza", "donut", "cake", "chair", "sofa",
                  "pottedplant", "bed", "diningtable", "toilet", "tvmonitor",
                  "laptop", "mouse", "remote", "keyboard", "cell phone",
                  "microwave", "oven", "toaster", "sink", "refrigerator",
                  "book", "clock", "vase", "scissors", "teddy bear",
                  "hair drier", "toothbrush"]

    while True:
        success, img = cap.read()
        results = model(img, stream=True)

        for r in results:
            boxes = r.boxes

            for box in boxes:
                # bounding box
                x1, y1, x2, y2 = box.xyxy[0]
                # convert to int values
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)

                # Display top-right corner coordinates
                top_right_text = f"({x2}, {y1})"
                cv2.putText(img, top_right_text, (x2, y1 - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

                # put box in cam
                cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 3)

                # confidence
                confidence = math.ceil((box.conf[0] * 100)) / 100
                print("Confidence --->", confidence)

                # class name
                cls = int(box.cls[0])
                print("Class name -->", classNames[cls])

                # object details
                org = [x1, y1]
                font = cv2.FONT_HERSHEY_SIMPLEX
                fontScale = 1
                color = (255, 0, 0)
                thickness = 2

                cv2.putText(img, classNames[cls], org, font, fontScale,
                                                        color, thickness)
        cv2.imshow('Webcam', img)
        if cv2.waitKey(1) == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

7.3 tracker.py

Listing 8: Euclidean Tracker Module

```
import math


class EuclideanDistTracker:
    def __init__(self):
        # Store the center positions of the objects
        self.center_points = {}
        # Keep the count of the IDs
        # each time a new object id detected, the count will increase by one
        self.id_count = 0

    def update(self, objects_rect):
        # Objets boxes and ids
        objects_boxes_ids = []

        # Get center point of the new object
        for rect in objects_rect:
            x, y, w, h = rect
            cx = (x + x + w) // 2
            cy = (y + y + h) // 2

            # Find out if that object was detected already
            same_object_detected = False
            for item_id, pt in self.center_points.items():
                dist = math.hypot(cx - pt[0], cy - pt[1])

                if dist < 25:
                    self.center_points[item_id] = (cx, cy)
                    # print(self.center_points)
                    objects_boxes_ids.append([x, y, w, h, item_id])
                    same_object_detected = True
                    break

            # New object is detected we assign the ID to that object
            if same_object_detected is False:
                self.center_points[self.id_count] = (cx, cy)
                objects_boxes_ids.append([x, y, w, h, self.id_count])
                self.id_count += 1

        # Clean the dictionary by center points to remove IDs not used anymore
        new_center_points = {}
        for object_boxes_id in objects_boxes_ids:
            _, _, _, _, object_id = object_boxes_id
            center = self.center_points[object_id]
            new_center_points[object_id] = center

        # Update dictionary with IDs not used removed
        self.center_points = new_center_points.copy()
        return objects_boxes_ids
```

In the *tracker.py* file there is the *EuclideanDistTracker* function which is used for tracking objects based on distance between their centroids. The constructor initializes two main attributes *self.center_points*, a dictionary to store the center points of detected objects and *self.id_count*, a counter to assign unique IDs to each newly detected object.

The update method is used to update the object tracker with new object data. It takes *object_rect* as an argument, which is a list of bounding box coordinates for each detected object in a frame. For each object, it is computed the center point by averaging its coordinates. The method then checks if any of these new objects match previously detected objects(based on a distance threshold). If a match is found (i.e., the object was already detected in a previous frame), the object retains its ID. If no match is found, it's considered a new object and assigned a new ID.

The matching is done using the Euclidean distance between the new object's center and existing center points. If the distance is less than a set threshold (here, 25 units), the objects are considered the same. After processing all objects, the method updates *self.center_points* to only include center

points of the currently active objects (those present in the current frame). The update method returns *objects_boxes_ids*, a list containing the bounding box coordinates and the assigned ID for each object.

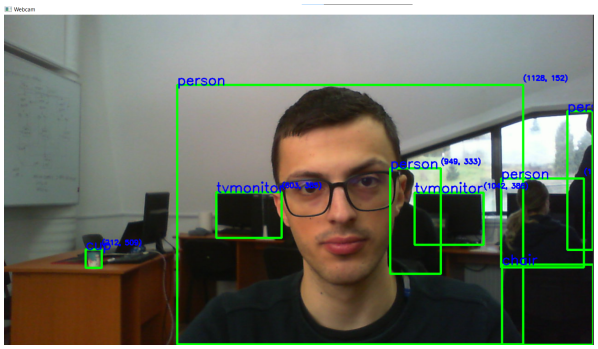
8 Testing and Validation

The tests for this project were done using a Microsoft LifeCam HD-3000 Webcamera. It has a resolution of 720 pixels so it is HD with a ration of 16:9. All the following testing and validation images were taken during the laboratory in the environment.

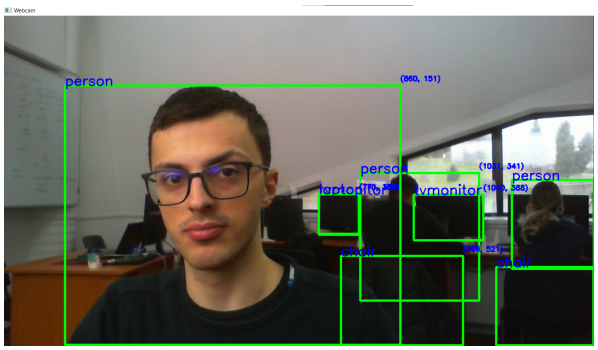
Figure 12: Microsoft LifeCam HD-3000

One can observe in Figure 13. that the program identifies some objects from the frame like person, laptop, chair, TV monitor or cup. The main person on the front is moving between the two frames and this is also told by the change of coordinates on the top-right corner of the bounding box.

Figure 13: Testing tracking and detection in lab



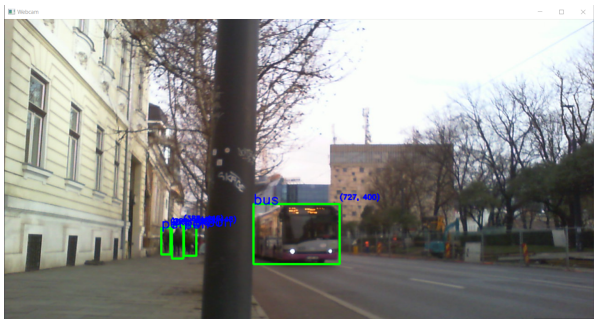
(a) First position of the person is (1128, 152)



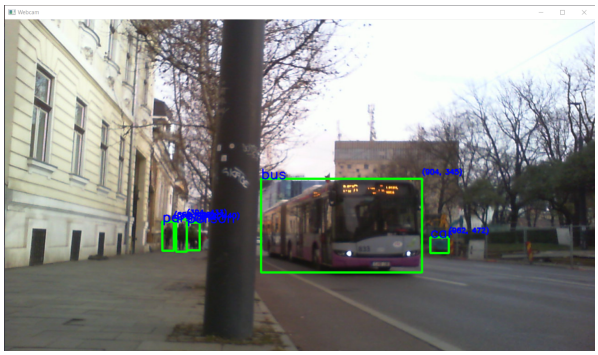
(b) Second position of the person is (860, 151)

Testing on a more complex environment was done on the Figures 14, 15 and 16.

Figure 14: Testing tracking and detection in city

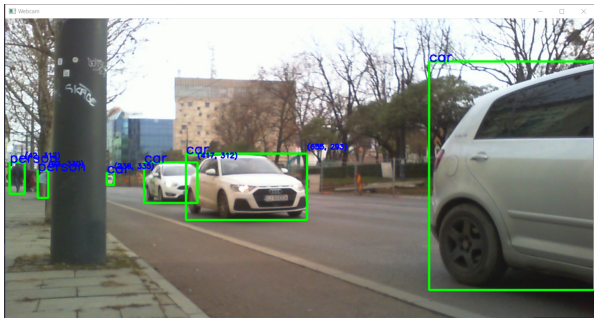


(a) First bus position on (727, 400)

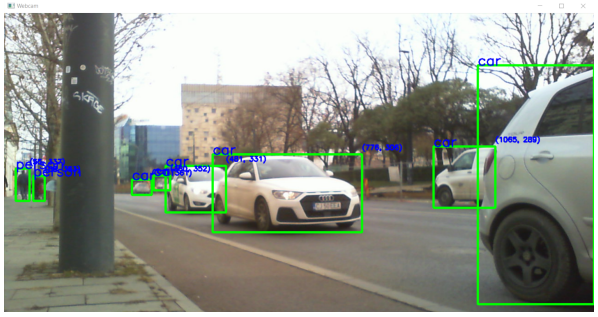


(b) Second bus position on (904, 345)

Figure 15: Testing tracking and detection in city

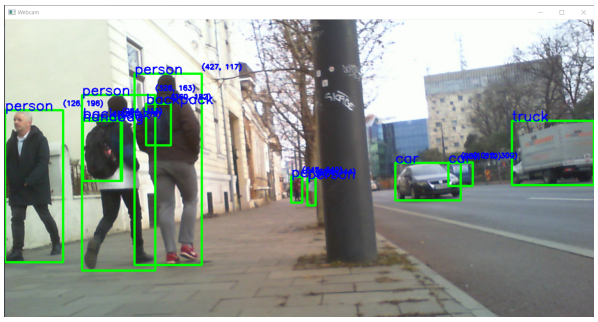


(a) First white car position on (655, 293)

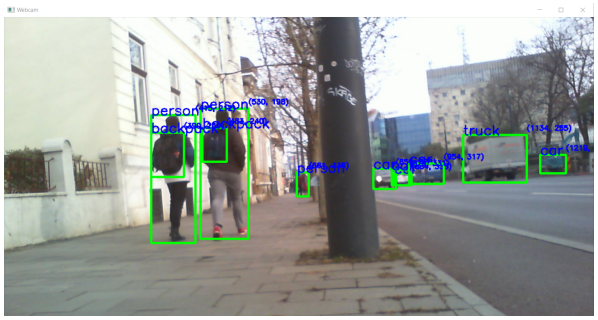


(b) Second white car position on (776, 306)

Figure 16: Testing tracking and detection in city



(a) First persons group position on (326, 163) and (427, 117)



(b) Second persons group position on (415, 212) and (530, 198)

The previous testing results are optimistic and demonstrate that using a YOLO model based approach to track and detect objects in moving could lead to a tool for achieving the given goal.

9 Conclusions

In summary, this project has successfully developed a robust method for detecting and tracking moving objects in real-time, utilizing a web camera and the powerful capabilities of Python's programming tools and computer vision techniques.

The system demonstrates high accuracy and efficiency in identifying moving objects across various environments, showcasing the effective integration of modern programming with advanced computer vision methodologies. This achievement not only fulfills the project's initial objectives but also sets a precedent for future innovations in this domain.

Moreover, the potential applications of this technology are vast and impactful. In the realm of security surveillance, this system can significantly enhance monitoring and threat detection capabilities, offering a more reliable and sophisticated approach to safety. Its applicability in traffic control systems opens avenues for smarter, data-driven decision-making, contributing to the enhancement of transportation safety and efficiency.

Additionally, the adaptability of this technology in interactive systems illustrates its versatility, making it a valuable tool in diverse fields. The successful execution of this project marks a significant advancement in the field of computer vision and object tracking, opening doors for further research and application in real-world scenarios.

References

[1] Chandrashekar. Object tracking explained with traditional computer vision and deep-learning based methods, 2023. https://vcs1994.medium.com/roadmap-of-object-tracking-fc520a0bae2a [Accessed: 27 Octomber 2023].

[2] Nico Kingler. Object tracking in computer vision 2023, 2023. https://viso.ai/deep-learning/object-tracking/ [Accessed: 20 October 2023].

[3] Rohit Kundu. Yolo: Algorithm for object detection explained, 2023. https://www.v7labs.com/blog/yolo-object-detection [Accessed: 25 Octomber 2023].

[4] Sovit Ranjan Rath. Moving object detection using frame differencing with opencv, 2023. https://debuggercafe.com/moving-object-detection-using-frame-differencing-with-opencv/ [Accessed: 11 October 2023].

[4] [2] [1] [3]